FWS
Flight Weather Station

Projektpraktikum

Johannes Kasberger
0616782

Markus Klein
0726101

Contents

List of Figures

Chapter 1

Introduction

1.1 About FWS

Chapter 2

Slave

Chapter 3

Master

3.1 About the Master

The master is written in Java and uses the jamod Library[1] for the communication with the sensors using the modbus protocol. The view is created with the SWT Library[2]. The description of the master is separated in several parts:

- View (see 3.2)
- Configuration (see 3.3)
- Stations (see 3.4)
- History (see 3.5)

Before the collection process of the measurement values can be started it's necessary to create a configuration. In this configuration all sensors are defined. A sensor must be added to the master before it can be used. Each sensor has an individual name and ip address. In this context it's referred as a station. In the configuration phase it's necessary to specify the kind of data the stations collect. The data that is transferred from the station to the master is called Input Parameter. To configure the sensor it's possible to define Configuration Parameters. These values are transferred from the master to the sensor. To map the parameters to the memory in the station they must be bound to addresses. This happens individually for each station. So it's possible to collect different data from different stations. It's also possible to configure the kind of plots that are generated (see 3.3).

[1] http://jamod.sourceforge.net/
[2] http://www.eclipse.org/swt/

After the configuration of the stations the collection process can be started. Each station is polled in its own thread. One collector thread collects the data from the station threads, generates a text file with the information about the current values from the stations and draws the plots. The time between the polling of the stations and between generation of the output files can also be configured. As soon as the current day changes the values from the last day are aggregated to a history value. The values from the station are kept for two days so it's possible to build a plot based on hourly data from the last day.

3.2 View

The view is created with the help of SWT. SWT uses the os drawing apis to draw the widgets. So the look and feel of the application is very good integrated in the os it runs in.

Screenshots

3.3 Configuration

The views that support the configuration are 3.3, 3.2,

The result of the configuration phase is an xml file with all settings in it. The location of this file is os dependent.

- **OS X**: /Users/{username}/Library/Application Support/FWSMaster/settings.xml
- **Linux**: ~/.fws_master/settings.xml
- **Windows**: C:\{User Dir} \.fws_master \settings.xml

See example in Listing 3.1

Listing 3.1: Sample settings file

```
<?xml version="1.0" encoding="UTF—8" standalone="no"?>
<fws_config>
<path>/home/user/FWSMaster/output</path>
<generatortime>5</generatortime>
<autostart>false</autostart>
<plotwidth>800</plotwidth>
<plotheight>600</plotheight>
```

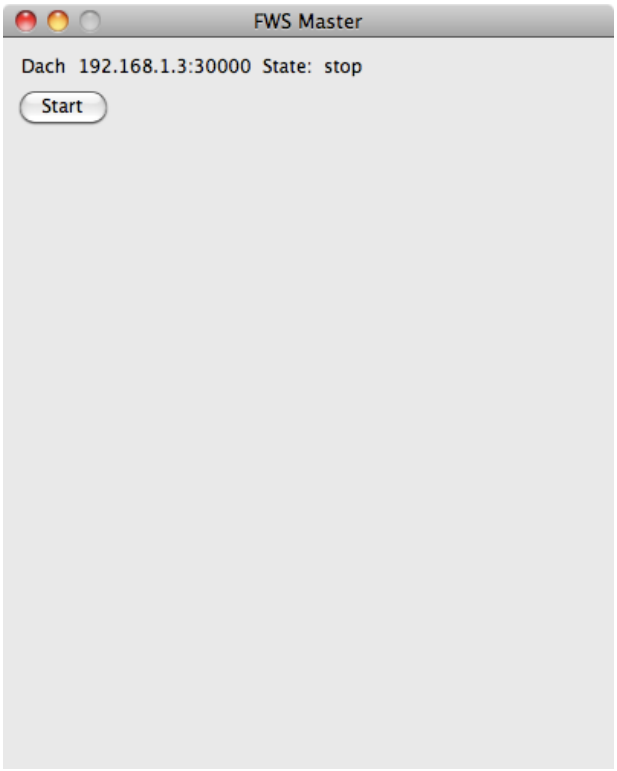

Figure 3.1: Quick overview about the stations and their status

```
<parameter typ="config">
<name>Messintervall</name>
</parameter>
<parameter typ="input">
<name>Temperatur</name>
<unit>°C</unit>
<format>6553.6</format>
<history>MAX</history>
</parameter>
<parameter typ="input">
<name>Windrichtung</name>
<unit>Richtung</unit>
<format>65536</format>
<history>AVG</history>
</parameter>
<station intervall="2" ip="192.168.1.3:30000" name="Dach">
```

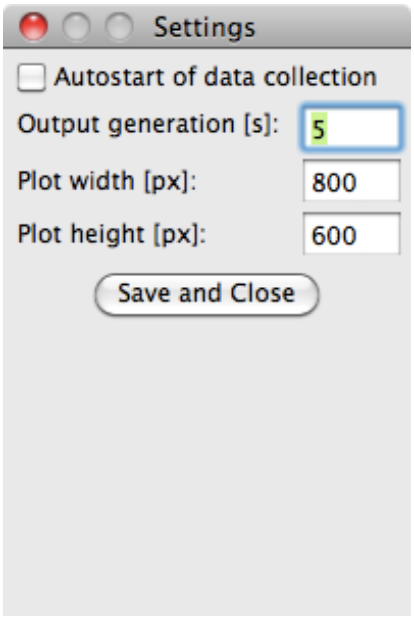

Figure 3.2: Configuration of the basic settings

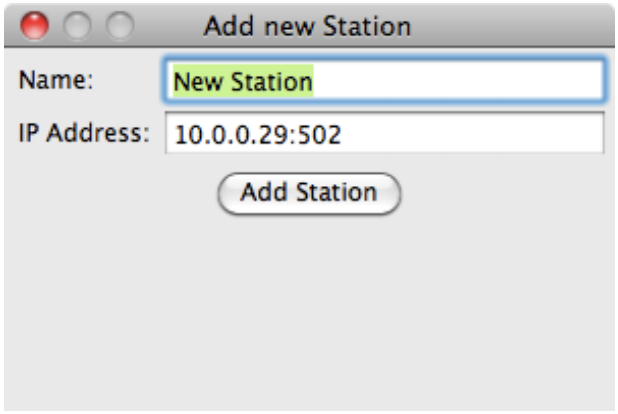

Figure 3.3: Adds a new station to the master

```
<binding active="true" address="3" parameter="Messintervall"
    transfered="false" type="config" value="1000"/>
<binding active="true" address="0" parameter="Temperatur" plotconfig="
    d4;1h24;" type="input"/>
<binding active="true" address="1" parameter="Windrichtung" plotconfig
    ="d4;c1;h24;" type="input"/>
</station>
```

```
</fws_config>
```

Parameters

There are two types of parameters. Input Parameters and Configuration Parameters. Each parameter has a unique name and a boolean value if it's enabled. The name is the identifier of the parameter so it has to be unique. Beside of these common attributes the Configuration and Input Parameters have further attributes. The parameters must be assigned to station addresses. This process is referred as binding a parameter to a station. To bind a parameter to a station, the user must define the address on the station where the parameter is saved.

Input Parameter

Additional attributes of an Input Parameter:

- Unit
- Format
- History Function

Unit The Unit is just used for the description in the generated files. Available units are:

- speed $\frac{m}{s}$
- speed $\frac{km}{h}$
- frequenzy Hz
- direction
- temperature $°C$

Format The transferred value from the station is a 16 bit integer value. To be able to display floating point numbers it's possible to set the output format to the desired form. Example: The temperature equals $23.3°C$. The station measures the temperature and converts it to 233. This integer value is transmitted to the master. The master converts the number back to the desired format. So this example would result in 23.3.

History Function There are three different history functions available. For more details about these functions and how they are used please read 3.5.

- average
- minimum
- maximum

Configuration Parameter

Additional attributes of a Configuration Parameter:

- value

Value A configuration parameter is a value that is transferred from the master to the slave. The value that is transferred is saved in the attribute value. This value must be an 16 Bit integer value.

Plots

The plots are generated with the help of the jFreeChart Library [3]. The plots can be configured for every binding of input parameters. Each binding can have several plots assigned. It's also possible to plot more than one data into one plot. The plots are configured with a string. The simplest configuration looks like `h24;` With that configuration one plot is generated and the data for this plot are the values from the last 24 hours. The plots are generated in the output directory that can be set by the user.

It's possible to use different data ranges for the plots. To allow the user to choose one range there are three different time bases available:

- c - current
- h - hours
- d - days

Current Use the newest values for the plot. Currently this is only implemented for the wind direction. See example in figure 3.4.

[3] http://www.jfree.org/jfreechart/

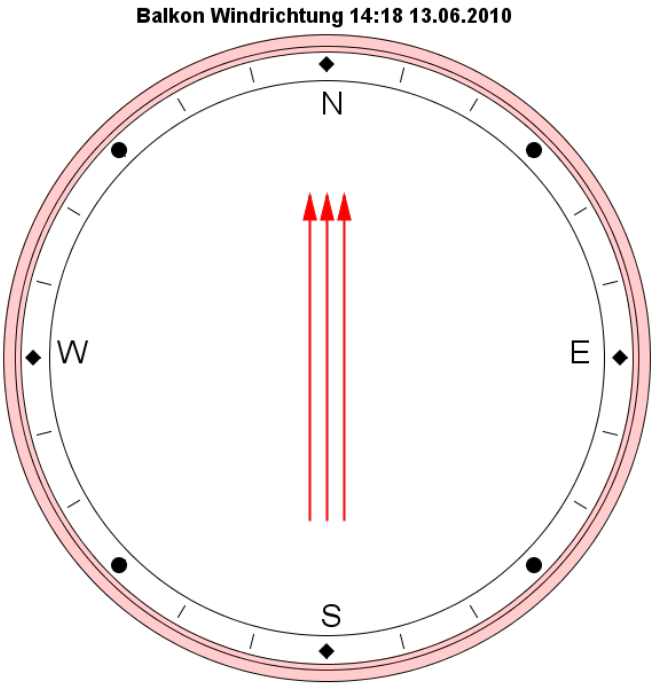

Figure 3.4: Current Wind direction plot

Hours Use the values of the last hours for the plot. The amount of the hours is specified in the plot configuration. It's the number after the timebase. Example of an 24 hour plot see figure 3.5.

Days Same plot as with the hours timebase but the values from the last days are used. For more details about the history see section 3.5.

Configuration Syntax

Each plot is specified by three different parts: [Number]CharacterNumber;

First Number: ID This number is optional and controls if more than one input parameter is drawn in one plot. All configured plots with the same number are drawn in the same plot.

Character: Timebase Defines the timebase. Explained in paragraphs above.

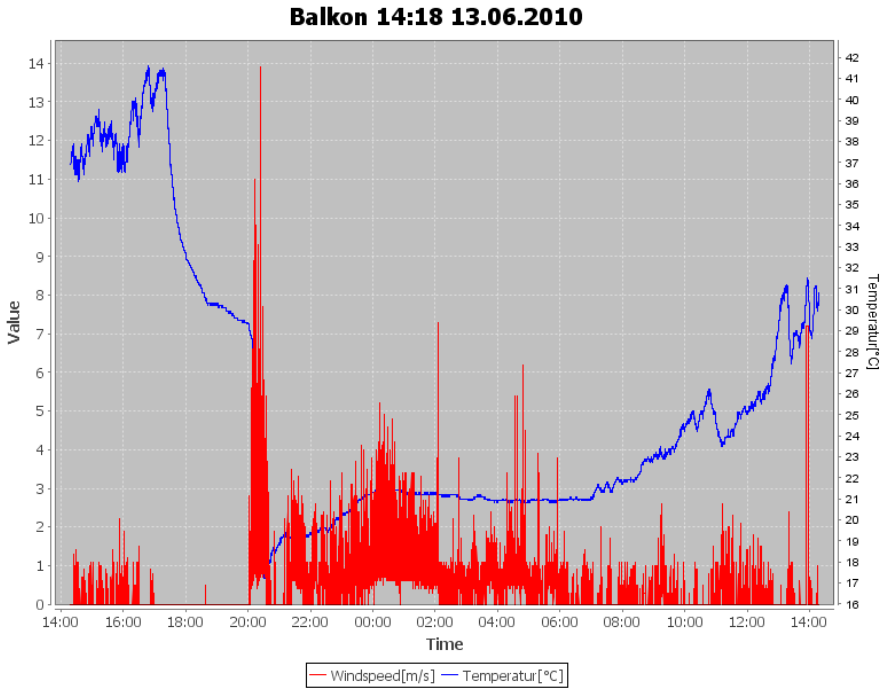

Figure 3.5: Last 24 hours of two separate parameters in one plot

Second Number: Amount of data The second number defines how much data will be in the plot. `d4;` will plot the last four days.

End of Configuration Each configuration must end with an `';'`.

Example For the configuration string `h24;1h24;d30;c1;d365;` following plots are generated:

- Last 24 hours
- Last 24 hours with other data with ID 1
- Last 30 days
- Current value
- Last 365 days

Detailed Information about the plots

If two different data sets should be plotted in one diagram there are two axis with an own scale generated. When more that two data sets should be drawn

in one diagram they share one axis. So it's advisable to generate more plots with two datasets each in it to keep the diagrams tidy.

The direction parameters values are not connected with a line. Otherwise the diagrams would be complicated if the wind direction changes a lot.

The direction parameter values are mapped to a description of the direction. So 0° result in N(orth) see figure 3.6

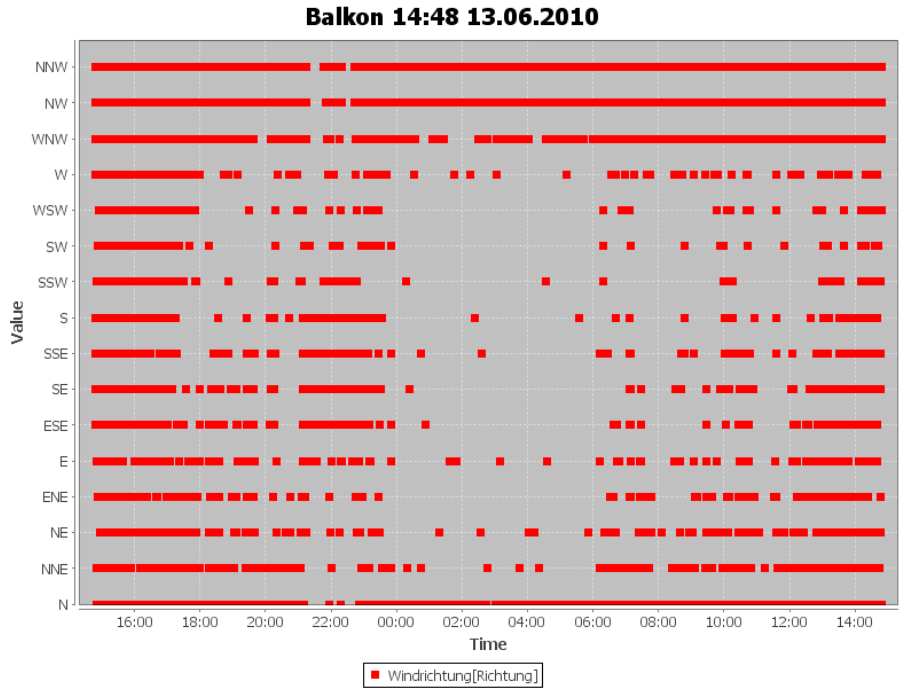

Figure 3.6: Axis description of direction parameter

3.4 Stations

After the stations are configured they are added to the station controller. This controller takes care for starting and pausing the station threads.

Transfering the Data

Each station runs in it's own thread. This thread polls in its own interval the data values from the station. For each transmission a new TCP connection is opened. After the value has been transferred the connection gets closed. Each station has a list of measurements. A measurement consists of

a timestamp and a value. These lists are collected from the Output Generation Thread. If a station is paused its thread gets suspended and waits for the wake up signal from the controller.

The Modbus function that is used to transfer the data is read register.

Transferring the configuration

When the user wants to upload a new configuration the station will call Write Register for each Configuration Parameter and reads back the answer of the station. If the answer equals the write register value the value is marked as transferred. A transferred value isn't uploaded again if it is unchanged.

Change the IP Address

The IP Address represents a special configuration parameter. It is always mapped to the address 0 and 1 on the station. These two 16 bit registers are used to save the ip address. When the ip address has changed it's transferred to the station. After the successful transmission of the two values the new ip is saved. Otherwise the old ip is kept.

Collecting the Data

A data collector thread runs in background an collects the data from all the stations. For each station/parameter combination an own list is kept with all the recent values in it. But it's not the normal measurement that is kept. The measurements get converted in a simpler data type that can be serialized. This datatype has no references to other classes.

In the collector thread the text file is generated that contains the information about the current status of the stations. Each station is represented in this file. The information written in this file are the current values of the Input Parameters bound to that station. This value is the average of the measurements since the last run of the collector. To be able to see whether the value changes the standard deviation is calculated and written to the output. For example result file see listing 3.2. For each parameter a new line is started in the result file. Syntax is `name[unit]:value;standard deviation;`

Listing 3.2: Sample result file

```
16:53:36 13.06.2010
Balkon
Windspeed[m/s]:0.0;0.0;
Temperatur[°C]:25.474999999999998;0.0452267016866652;
Windrichtung[Richtung]:232.5;138.14518054963375;
```

```
====
Dach
Windspeed[m/s]:0.0;0.0;
Temperatur[°C]:26.672999999999998;0.03345864;
Windrichtung[Richtung]:232.5;138.14518054963375;
====
```

3.5 History

The collector adds the measurements to the history controller. This controller converts the values to the entries in the history. Each input parameter has two histories. One short term history and one long term history.

Short term history

In the short term history there are the history entries from the last two days. This history is queried when plotting a diagram with the 'h' timebase. To this list all new measurements are added as soon as new data arrive from the collector. During this phase it's checked whether the day has changed since the last time new measurements have been added. If that is the case the day is transferred to the long term history. There are at least the last 24 hours in the short term history.

Long term history

As soon as a new day has started the past day gets transferred to the long term history. To avoid to much values in this list one representing value is calculated for the day. This happens with one of the history functions listed in 3.3. During the calculation of the history value all measurements older than the last day are removed from the short day history. The representing value is added to the long term history.

Storage of the history

The short and long term history are serialized to the hard disk. After new data have been added to the history it's saved to the hard disk. To prevent the user closing the master during the I/O operations a semaphore is used. [4]

[4] Without that semaphore we often had corrupted history files because the master closed at a critical instant.

An old history is kept to prevent loosing the history if the master crashes during writing the history. Before saving the history the thread locks the semaphore. After that it renames the current history to another filename. After that the new history is saved. The history is saved now so the semaphore is released.

If an exception encounters during loading the history the master tries to load the old history file. If that also results in an exception a new history is created.

3.6 Implementation details

The source code can be downloaded at github[5]. In the doc folder there is the javadoc generated source code documentation.

In the following list you see which classes implement which functions. For a class diagram see figure 3.7.

- View: All classes that start with view
- Configuration: Parameter, InputParameter, ConfigParameter, StationInputBinding, StationConfigBinding, Station
- Save the configuration: PersitencePreferences, all classes with contentenhandler in its name
- Data collection: Station, MeasurementCollector, Measurement, StationInputBinding, MeasurementHistoryController, MeasurementHistory, MeasurementHistoryEntry
- Plotting and Output generation: MeasurementCollector, PlotBase, TimePlot, CurrentPlot
- ModBus: ModbusWrapper, Station

[5] http://www.github.com/schugabe/fws

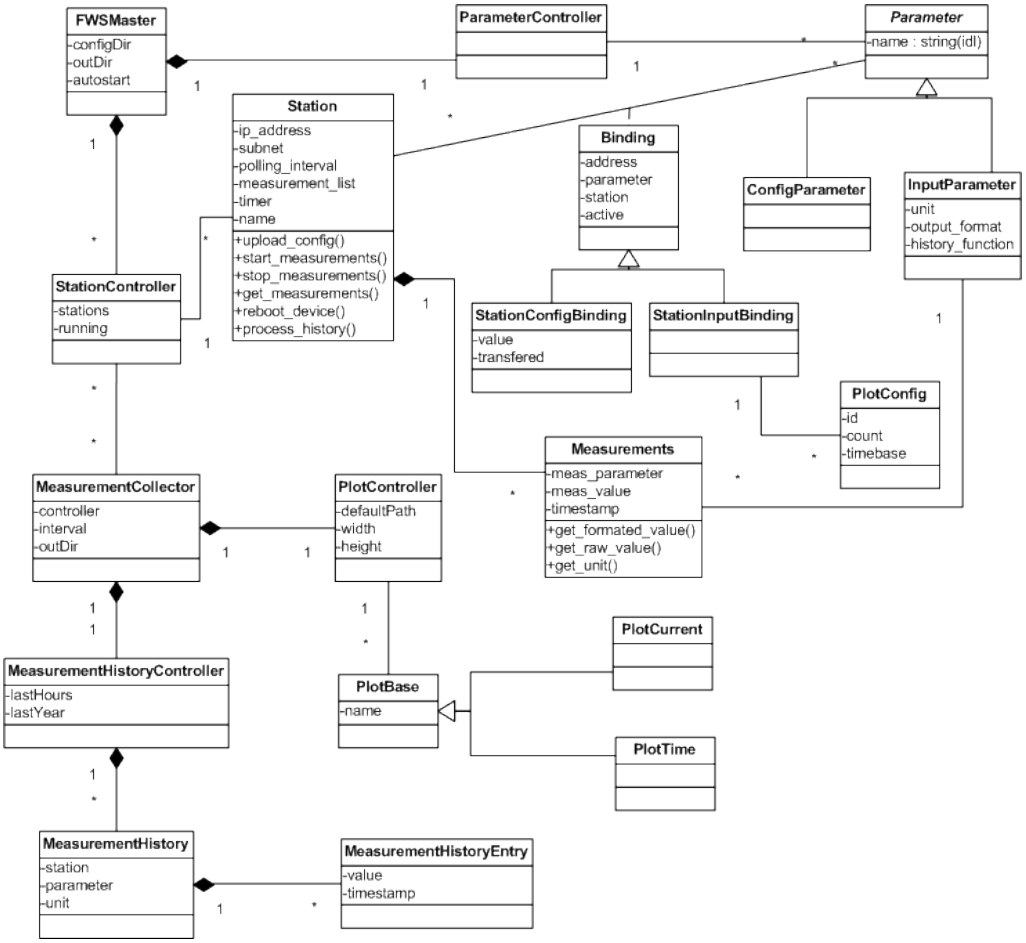

Figure 3.7: Class diagram of the master