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 Java and uses the jamod Library[1] for the communication with the sensors over the modbus protocol. The view is created with the SWT Library[2]. It's separated in several parts:

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

It's needed to create a configuration file before collection process of the measurement values could be started. A sensor must be added to the master before it can be used. Each sensor has a individual name and ip address and it's referred as a station. In the configuration phase it's necessary to specify the kind of data that 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).

After the configuration of the stations the collection process could be started. Each station is polled in its own thread. One collector thread collects the data from all stations, generates a text file with the information

[1] http://jamod.sourceforge.net/

[2] http://www.eclipse.org/swt/

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 the current day changes the values from the last day are aggregated to a history value. This daily history is kept for one year. The values from the station are kept for two days so it's possible to plot hourly data.

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

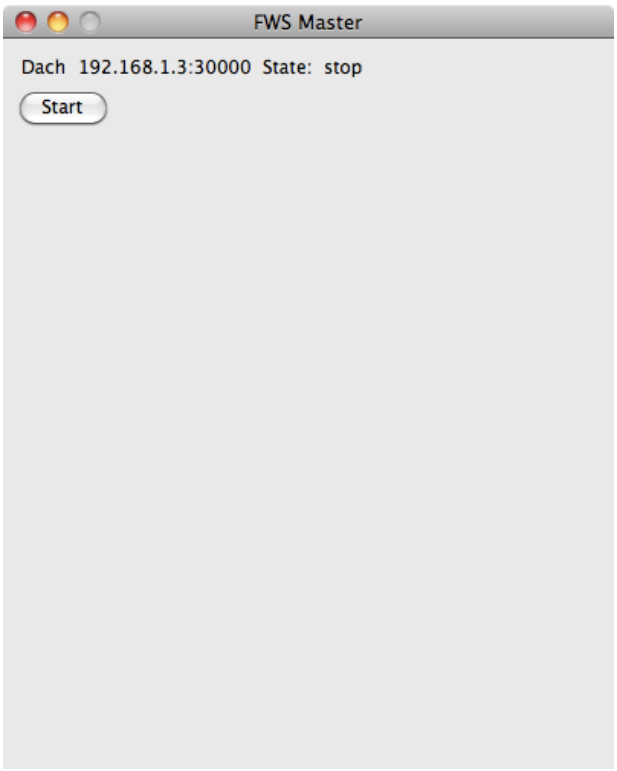

Figure 3.1: Quick overview about the stations and their status

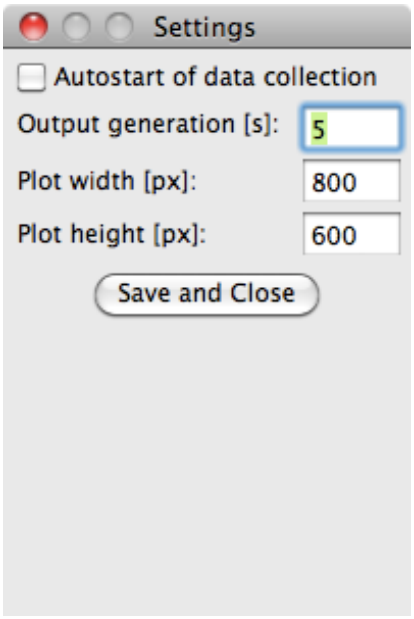

Figure 3.2: Configuration of the basic settings

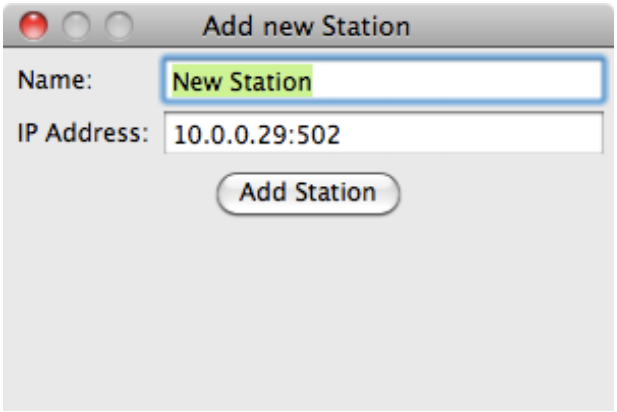

Figure 3.3: Adds a new station to the master

3.3 Configuration

The necessary views for the configuration are 3.3, 3.2,

The result of the configuration phase is an xml file with all settings in it. 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>
<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">
<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 parameters has a unique name and a boolean value if it's enabled. The name is the identifier of the parameters so it has to be unique. Beside of these common attributes the configuration and input parameters have further attributes.

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 to have an integer value. On the master the output format is set to the format with one digit before the decimal point. When such an input parameter is used it's converted back to 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. 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 C 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 from the user.

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

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

Current Use the last values for the plot. Currently this is only implemented for the wind direction. Results in an plot like Figure 3.4.

Hours Use the last 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 chapter 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.

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

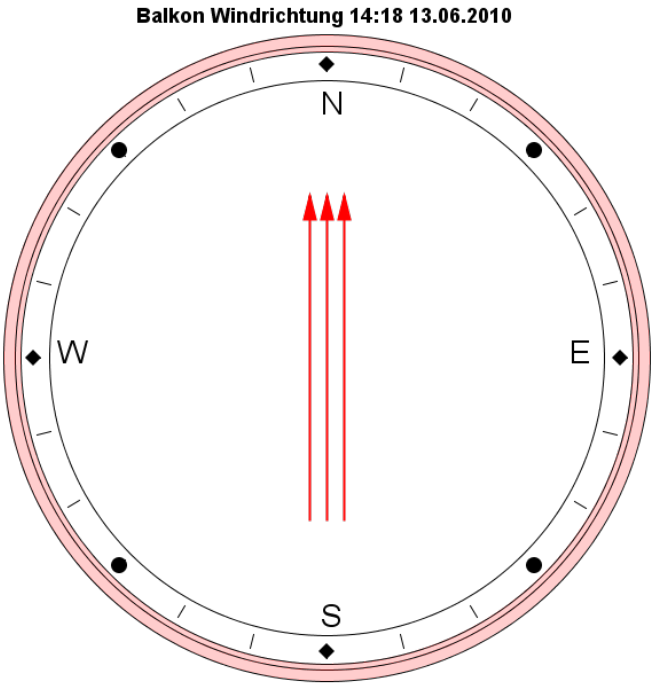

Figure 3.4: Current Wind direction plot

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

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

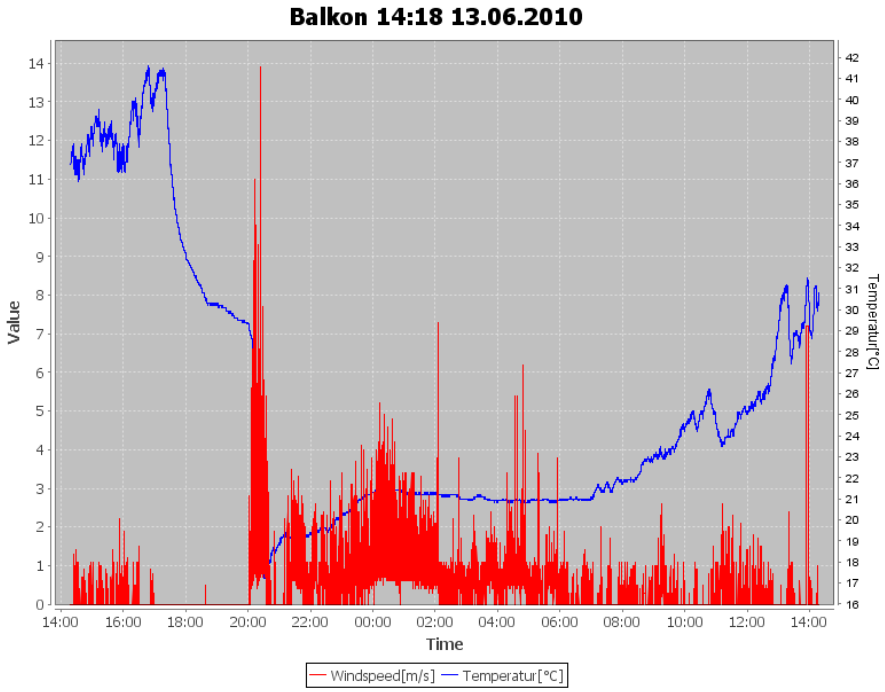

Figure 3.5: Last 24 hours of two separate parameters in one plot

Detailed Information about the plots

If two different data sets should be plotted in one plot there two axis with own scale generated. When more that three data sets should be drawn in one plot they share one axis. So it's advisable to generate more plots with two datasets each in it to keep the plots tidy.

The direction parameters values are not connected with a line. Otherwise the plot would be unreadable if the wind directions 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

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.

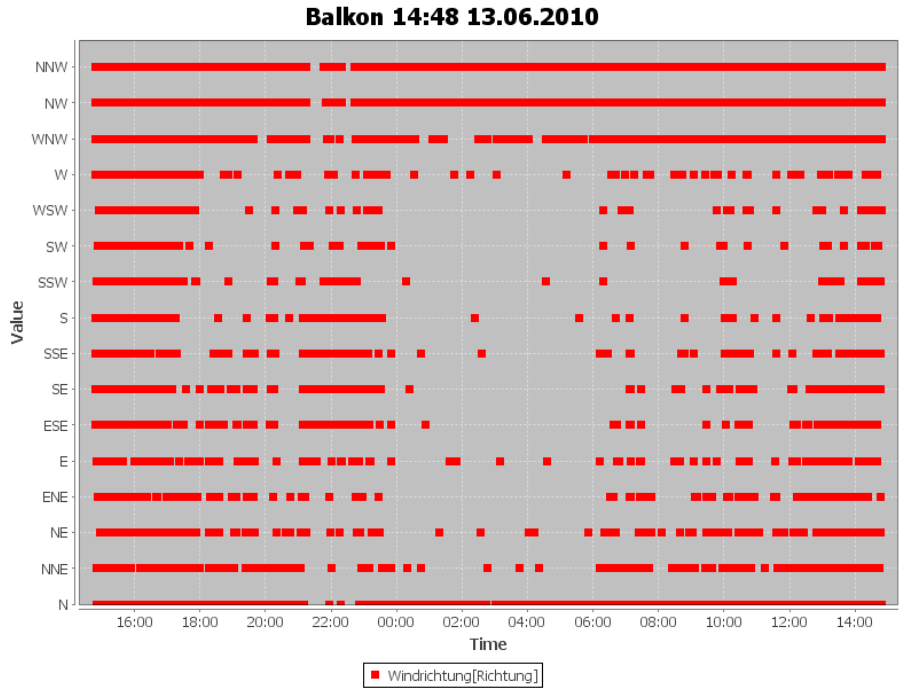

Figure 3.6: Axis description of direction parameter

Getting the Data

Each station has it's own thread in that it polls in its own interval the data values from the station. For transmission a new tcp connection is opened. After the value has been transferred the connection is closed. Each station has a list of measurements. A measurements consists of an timestamp and a value. If a station is paused its thread is 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 several times and reads back the answer of the stations. 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 a 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 an 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. For each station is an entry in this file. This entry contains the current value of each parameter bound to that station. This value is the average of the measurements since the last run of the collector. To see if 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 are the history entries from the last two days. This is history is queried when plotting a plot with the 'h' timebase. To this list all new measurements are added as soon new data arrives from the collector. During this phase it's checked if 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

Once a new day has started the past day gets transferred to the long term history. To don't get to much values 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 saved to the hard disk with the help of serialization. After new data has been added to the history it's saved to the hard disk. To prevent the user from closing the master in this moment a semaphore is used to prevent closing during the I/O operations haven't finished. Without that semaphore we often had corrupted history files because the master closed at an critical instant.

To prevent from loosing the history when the master crashes during writing the history an old history is kept. When save history is called the thread locks the semaphore. After that it renames the current history to the old history. After that the new history is saved.

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

3.6 Implementation details

The source code can be downloaded at github[4]. In the doc folder there is the javadoc generated source code documentation.

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

To help you to get an overview over the source code here is an list with the function described in this document and in what classes it's implemented. There is also an class diagram 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

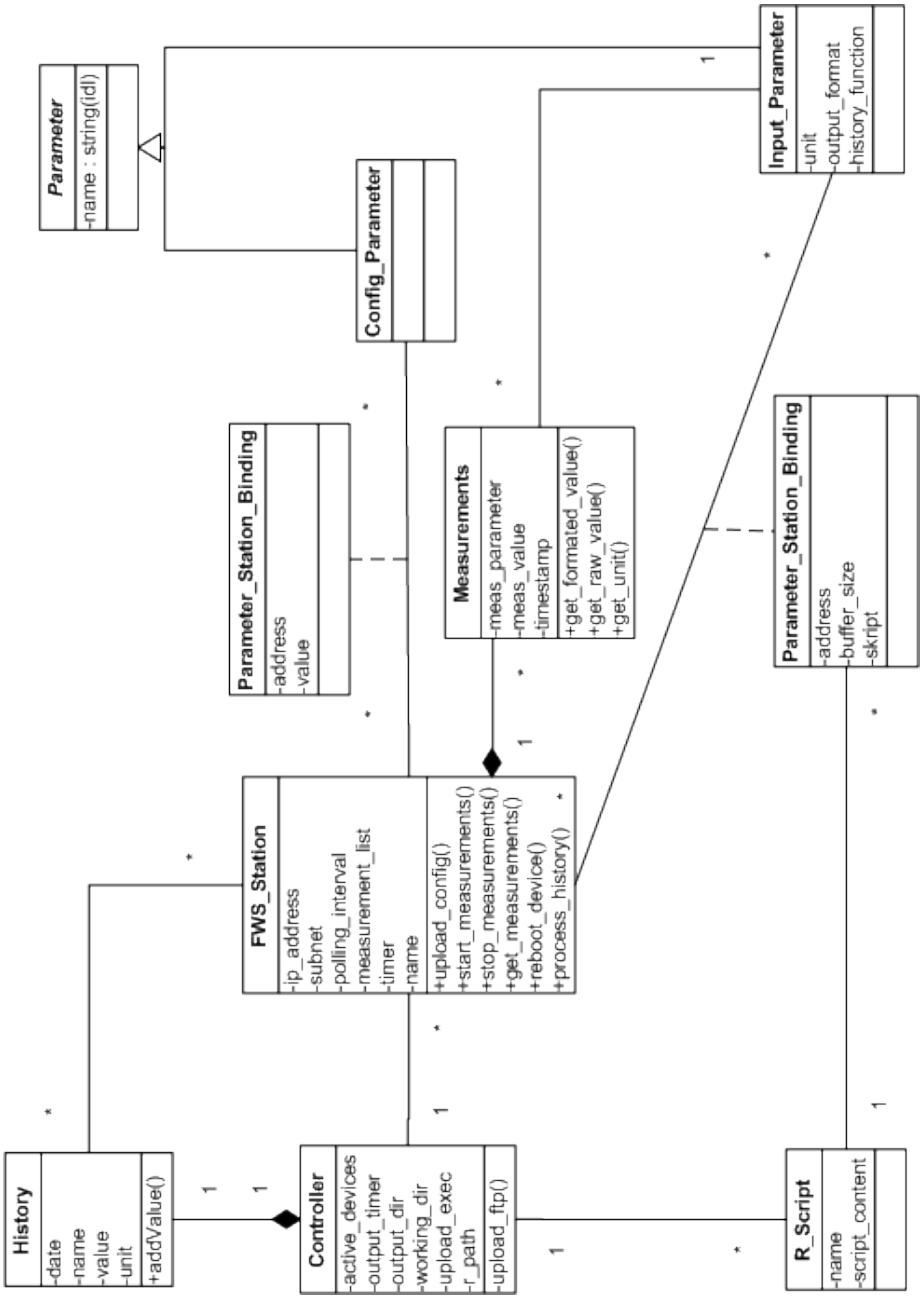

Figure 3.7: Class diagram of the master