Implementation of a platform independent client software for the GO Bluebox System

Nils T. Kannengießer
and
Thomas Ladehoff
Faculty of Computer Science and
Electrical Engineering
Kiel University of Applied Sciences
Kiel, Germany

Thorsten Knutz
GO Systemelektronik
Kiel, Germany

Helmut Dispert
Faculty of Computer Science and
Electrical Engineering
Kiel University of Applied Sciences
Kiel, Germany

***Abstract*—The BlueBox is a dedicated data acquisition system produced and commercialized by the North German SME GO Systemelektronik[1]. GO develops innovative systems for industrial measurement and control applications, starting out with individual single sensor solutions and expanding to the BlueBox, a complex general purpose system. Typical applications for the BlueBox can be found in the area of environmental monitoring, e.g. in the aquaculture industry. In this paper we discuss a software-based extension and interface designed for the BlueBox. The software, developed in Java, allows control of the BlueBox as well as reading and displaying BlueBox-acquired sensor data.**

Keywords: Data Acquisition, Sensors, Actuators, Java Technology, Java Applets.

I. Introduction

Modern data acquisition systems are strongly based on microcontroller systems connected to a variety of sensors and – to allow full control – to actuators. Whereas the microcontroller systems are in most cases relying on standard architectures and industrially available products, the necessary sensor/actuator units as well as the software are often custom-made.

GO Systemelektronik, an SME based in Kiel, Northern Germany, has been dedicated to the development of complex and innovative data acquisition systems, greatly implementing sophisticated sensor and when required actuator systems. Their major field of application is environmental monitoring with a strong emphasis on aquacultural systems used world-wide.

Although the BlueBox (as the master module for all connected slave ssensors) provides several interfacing and communication channels (e.g. using CAN bus), a generic way for using the systems over Internet-based channels has been missing. The existing legacy system is MS Windows based.

In this paper we present a new software system that has been developed under the requirement of being totally platform independent, allowing the display of BlueBox-based sensor data. Thus the way will be opened for generic monitoring and data acquisition systems.

Considering the current BlueBox system design based on standard windows software, it was decided to develop the new program package using Java Technology, which automatically leads to a platform independent design, considering that Java has been developed with that goal in mind. Java itself can be employed in different ways to develop and apply platform independent software.

For this R&D-project is has been decided to use the Java Applet technology. Java Applets have been around since the first realization of the Java environment. Recently a revitalization of the Applet philosophy could be observed.

The BlueBox system software incorporates its own webserver. Therefore the Java Applet can be easily accessed and loaded into the remote client systems.

II. Background

The BlueBox provides its own services, which are available through TCP/IP to every network or internet client.

On Port 14111 (TCP) the box is waiting on its command shell so that it is easily possible to get some sensor information by using terminal software like HyperTerminal or Telnet.

[1] http://www.go-sys.de

The applet software is mainly created for viewing sensor data. Therefore only commands for receiving information are used in this project. Control and management functions are not implemented in the current version presented in this paper..

In order to create this application within a reasonable amount of time the applet uses the libraries jFreechart and Jcalendar, which are both licensed under LGPL.

A. Structure of the applet

The applet is divided into three layers (Java packages) to achieve a clear logical separation between the different program parts.

- **blueboxapplet.gui**
  Implementation of the Graphical User Interface.
- **blueboxapplet.domain**
  Implementation of the processing tasks (e.g. analysis of sensor data) and the communication with the BlueBox
- **blueboxapplet.common**
  Classes that are used in the other layers.

Connection to the Bluebox

The connection between Bluebox and applet is based on sockets using the TCP/IP protocol.

getserialno	BlueBox serial number
getstarttime	Get database starttime(GMT)
getsensorno	Number of sensors
resetdam	Set DAM pointer to first DAM
getdam	Show next DAM info
resetsensor	Set sensor pointer to first sensor
getsensor	Show next sensor info
getsensordata	Show sensor data
getadamnr	Number of actuator DAMs
getadam	Show next actuator DAM info
getactuatornr	Number of actuators
getactuator	Show next actuator
gettime	Get BlueBox date & time
getposition	Get GPS or GEO position
getstatus	Get BlueBox status information
gettimeserver	Get NTP status
password(pw)	If password is required
gdb(…)	Get database entries
getchangelog(..)	Get changelog entries
quit	Close connection

Table 1 - BlueBox commands (shorted) [2]

The connection establishment and all following communication are encapsulated in the class *BlueboxCommunicator* of the package *blueboxapplet.domain*. The host and port of the Bluebox server are passed to the constructor. A second constructor is provided, which determines the host automatically.

After object creation the connection is established by calling the method initConnection which creates the socket.

The requests and responses are represented by strings sent over the network (for available request commands see table 1).

B. Receiving and processing the data

Sending and receiving data to and from the server is done by a generic method, named *getServerResponse*. It takes the request string as an argument and returns the received data in a list of strings (Java class *ArrayList*). Generally each element in the list represents the string of one data packet received from the server.

To determine the end of an incoming data stream, the received string is checked for a certain end condition (i.e. a certain string contained in the data). Therefore the method takes a further parameter.

This method *getServerResponse* is called by the methods, which analyze the data of the server:

- **getSensorList**
  Receiving the list of sensors from the database
- **gdb**
  Receive the data of a sensor (same name as the Bluebox command)

The most complex parts of these methods are processing the data.

In the method *getSensorList* one sensor after the other is read in a loop, because the server sends only the data of one sensor per request (request string *getsensor*).

The strings coming from the server get analyzed and for each sensor a new *Sensor* object is created. These objects are added to a list that is returned to the caller.

The method *gdb* takes the sensor and the start and end time as input parameters and builds the request string with this information.

The received data is saved in a list of strings (according to the return value of method *getServerResponse*). Each list entry contains several lines, which represent the database entries. The data in every line has to be extracted, which is done by the help of a regular expression.

The values of the lines are saved and put into a *SensorData* object, which gets also the timestamp of the first received database entry. Afterwards this object is returned.

The data produced by *getSensorList* and *gdb* is used by the GUI to build the selection of the sensors and the chart displaying the data of the chosen sensor.

C. The GUI

The GUI is divided into two parts. On the left side there are the options and controls and on the

right side the generated diagram (or nothing at start-up).

The left part is created by using the Netbeans Form Editor for easy adjustment of controls in further applications. The sensor names for the drop down-box are generated just in time, when starting the applet by fetching the sensor list from the server.

The interface itself is currently in German. After selecting the proper values a short check routine is looking for a correct period and chosen values before passing the values to the *gdb* method.

After receiving the data by the *gdb* method it is transferred directly to the classes of the jFreechart library, which are responsible for generating the diagram.

jFreechart has a lot of options like zooming or saving the current view as an image which was one of the reasons for using it.

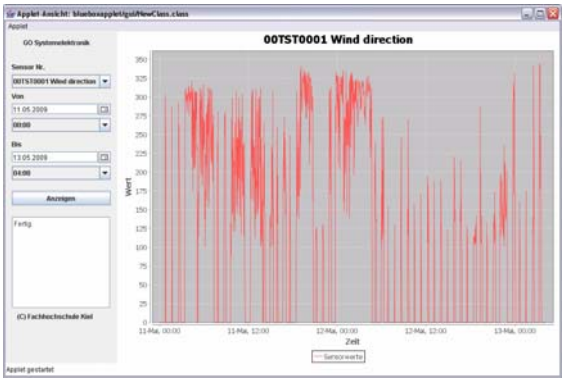

Figure 1 – Complete Form for application

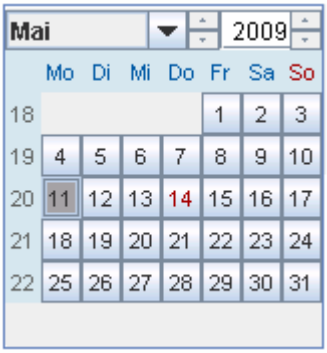

Figure 2 - jCalendar for choosing the date

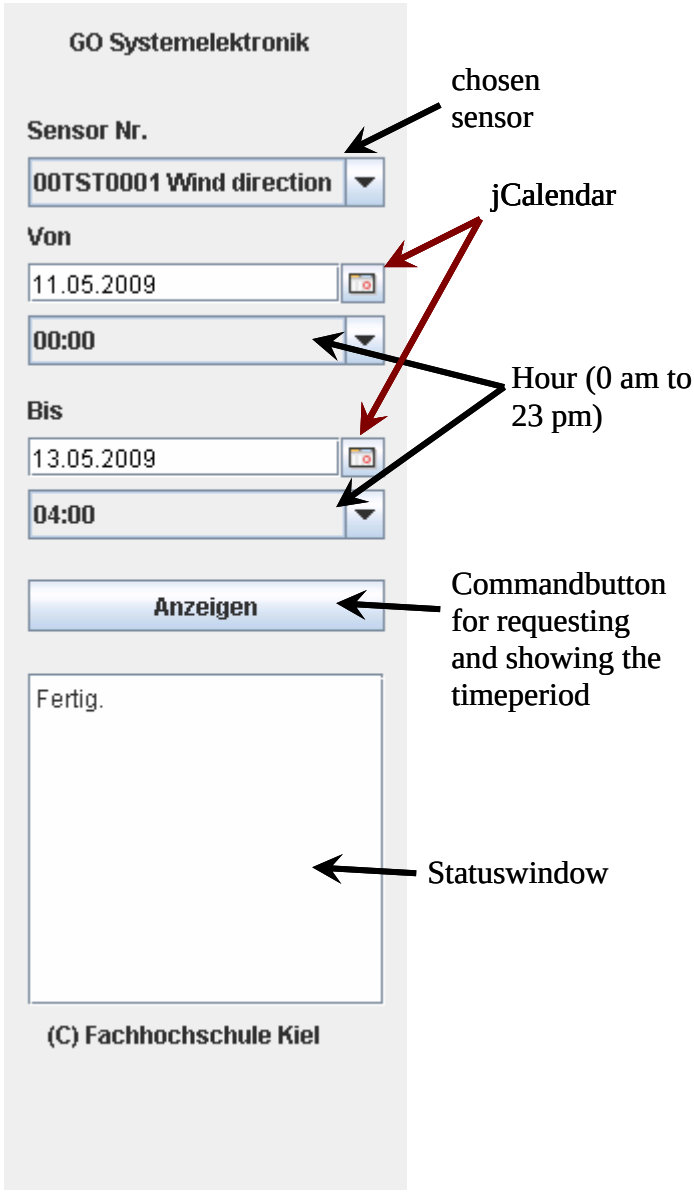

Figure 3 - Command interface

III. Conclusion

As Java is platform independent we created an easy to use applet in addition to the existing windows application. Our approach is that this Java applet is also 100% usable under Linux, MacOS and other operating systems, which support Java.

References

[1] Eric Roberts, "Resurrecting the Applet Paradigm", Proceedings of the 38th SIGCSE technical symposium on Computer science education, ACM, March 2007, pp. 521-525.

[2] GO Systemelektronik, Germany
www.go-sys.de

[3] jCalendar
www.toedter.com/en/jcalendar/index.html

[4] jFreechart
www.jfree.org/jfreechart