GENERAL PURPOSE DATA MONITORING SOFTWARE FOR PLATFORM INDEPENDENT REMOTE VISUALIZATION

Andreas Gschwandtner[(a)], Michael Bogner [(b)], Franz Wiesinger[(c)], Martin Schwarzbauer [(d)]

[(a, b, c, d)]Upper Austria University of Applied Sciences, Hagenberg Austria,
Hardware/Software Design & Embedded Systems Design

[(a)]andreas.gschwandtner@fh-hagenberg.at, [(b)]michael.bogner@fh-hagenberg.at,
[(c)]franz.wiesinger@fh-hagenberg.at, [(d)]martin.schwarzbauer@fh-hagenberg.at

ABSTRACT

This paper presents a novel general purpose data monitoring software for remote visualization. This tool can be used broadly in practical applications to record, trace, and display measurement information from any data source. It covers a wide range of use cases through the support of various chart types such as line graphs, pie charts, and bar graphs. Although designed as general purpose monitoring tool, it was specially intended for dedicated applications as embedded devices with limited resources with possibly no graphical output interface. The underlying idea is to develop a tool that can process data from any device, completely independent from its hardware, operating system, or software. Next to data monitoring, it can be used as central recording platform where the data will be evaluated later. This feature is especially useful during the development and debugging phase. Measurements, in particular erroneous values, can be easily detected using visual support.

Keywords: visualization, embedded system, remote, network

1. INTRODUCTION

"A picture is worth a thousand words" (Barnard, F., 1927) is an often used phrase that emerged in the USA in the early part of the 20th century. The meaning of it is that a picture tells a story just as well as a large amount of descriptive text. The advantages of information visualization compared to information description are as follows (Colin Ware, 2004):

1. Images can provide simultaneous information.
2. Images have expressions that are linguistically hard to describe.
3. Images can condense information very strong.
4. Images can easily identify or focus on important information.

The fictive example in Fig. 1 shall depict the difference in data visualization opposing a diagram to a tabular presentation form. The introduced example consists of a random voltage profile of an electronic component recorded by an embedded system. On the left side of the figure we can see the tabular presentation of the individual voltage levels over time. On the right, the same information is presented, but in difference visually processed (Otto-von-Guericke, 2008). In fact, the line diagram is much easier to comprehend.

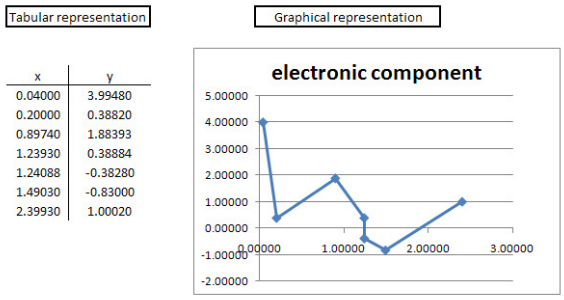

Figure 1: Different types of data visualization; tabular presentation (left), diagram presentation (right).

Due the fact, that human beings are more familiar with visually processed information, the line chart in Fig. 1 is the preferred method to turn the data into information. Furthermore, information has nothing to do with the amount of data; it is all about the appropriate representation (Schwarz, 2008). The main aim of this paper is the development of a platform independent visualization tool for displaying any kind of data. It can capture continuous, discrete, and instantaneous values. In order to be a general purpose tool, the supported chart types can be freely configured by the data source. The DataMonitor is not only tied to technical issues, but also generally usable. There exists no limitation of the visualization, which corresponds to semantics or meaning of data. This flexibility can be practical in many other fields, e.g.:

1. Automotive development (motor speed, exhaust gas temperature, torque measurements, system diagnostic, oil pressure...)
2. Chemistry and Biology
3. Geography (elevation profile)

4. Weather information (water level standings, CO2 emissions, temperature patterns...)
5. Home Automation (heating curves, heating duration…)
6. Financial economic (stock data)

2. COMMUNICATION

In computing, inter-process communication (IPC) is a set of techniques for the exchange of data among multiple threads in one or more processes. These processes may also run on different devices connected by a network. Due to the fact that we cover a large scope of applications, the platform independency is an important requirement of the visualization tool. The main objective was to support local and remote devices across heterogeneous platforms. Therefore we will basically focus on remote IPC techniques (Koch, 1993). After deeper investigations on different communication techniques (Microsoft Corporation, 2005), four main requirements were figured out. Tab. 1 shows an overview of the results of the elaboration of remote IPC methods.

Table 1: Rating of communication methods.

criteria	Remote procedure call	Named Pipe	Socket
flexibility	-	X	X
duplex operation	X	X	X
Connection-orientated	X	X	X
platform independent	-	-	X

Recently, to meet the required independency of the platform, the socket communication is prioritized. In particular, the socket communication (BSDZone, 2008) meets all criteria as shown in Tab.1. However, the software can be extended by other communications such as serial (RS232) or USB interfaces. In difference to common workstations where the serial port has largely been replaced by USB, embedded systems still count on this interface because of the low resource consumption. The advantage of the DataMontitor is the extensibility for other communication devices to support especially limited devices with different kinds of communication ports.

3. VISUALIZATION

Data visualization is the study of the visual representation of data. The main goal of data visualization is to communicate information in a clear and effective way through graphical means (Gregory, Hagen, Müller, 1997). It helps to provide insights into a rather sparse and complex data set by communicating its key aspects in a more intuitive way. With the support of graphical illustration of data, key-aspects can be filtered, meaning a simplification to the user. In general the different representations can be divided into conventional and innovative techniques:

1. Conventional techniques of data visualization are graph-based diagrams (as tree diagrams and flowcharts), chart types (as pie charts, beam diagrams and line diagrams) and many other variations and types of diagrams (e.g. exploded view, density map). Fig. 2 and Fig. 3 show three chart-like diagrams and an analog display. These kinds of diagrams display the relationship between two variables that take either discrete continuous or instantaneous values.
2. Innovative techniques are often in conjunction with scientific visualization and used for specific applications.

The fundamental idea of this work was to create a general purpose tool. In fact, it is focused on conventional presentation techniques.

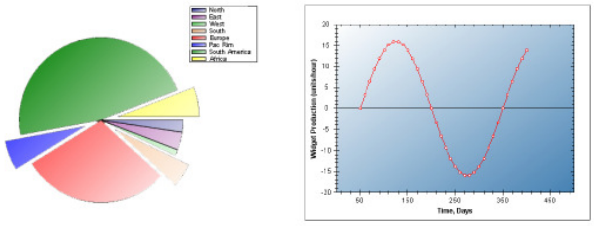

Figure 2: pie chart (left), line chart (right)

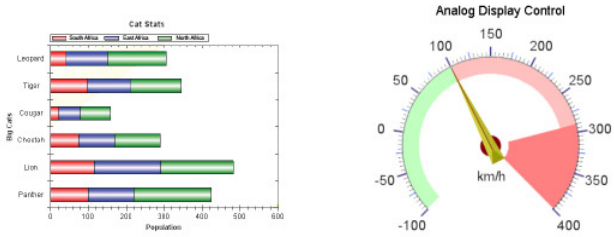

Figure 3: bar chart (left), analog display (right)

The DataMonitor supports commonly used chart-like diagrams and analog/digital display controls. In terms of flexibility, the tool can be easily extended to support further charts and remains backward compatibility. Because of the modular design concept, specific chart types can be easily integrated into the system. Moreover it doesn't interfere with the existing applications.

4. DESIGN AND IMPLEMENTATION

The software was designed as client/server system to assist in the creation of a platform independent system. The design consists essentially of two parts: the communication and the visualization.

4.1. Communication

To meet the requirements of an easy-to-use application, we use an ordinary communication protocol without request acknowledgments. The communication protocol implemented in this design is a TLV-protocol (Type, Length, and Value) to fulfill the communication and the data transfer. On the one hand, the TLV is a simple and sparse protocol, but on the other hand it allows the creation of a very flexible design. Because of the modular design of the TLV, we can integrate new features in the packet by using the value part of it, where another TLV packet can be inserted. Furthermore, we can generate new keywords (e.g. for specific charts) to extend the functionality and still remain backward compatibility.

In order to keep the effort for configuration at a minimum, there were only a few keywords defined for setting up a communication channel between the server and the client. This makes the DataMonitor applicable especially for limited devices with low memory and restricted communication resources. As described before, the server doesn't send an acknowledgment after a request at all. If the communication protocol was misinterpreted by the data source, the data sink will discard incoming data. In figure 4, the directed stream (one way) from the data source to the data sink can be seen. As the Internet Protocol (IP) defines a standard network byte order used for numeric values, we intend to use the same Big Endian format too. By default, the software supports socket communication over TCP/IP, but to be backward compatible to earlier software applications of the "Upper Austria University of Applied Sciences" the tool also supports the remote procedure call communication (RPC). The application is capable to switch the communication on the fly without the need to restart the program. The integration of other communication channels as RS232 or USB is described later in the implantation section.

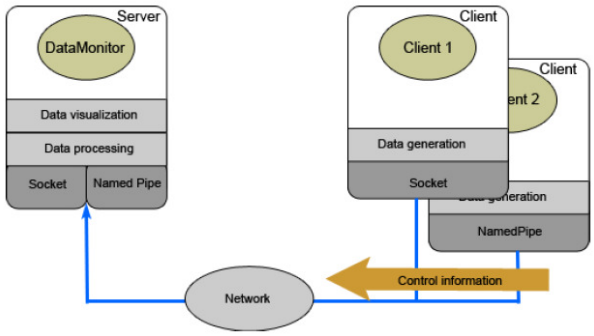

Figure 4: The DataMonitor mainly supports communication either via sockets or via RPC.

Next to the well-defined and easy-to-use protocol, we developed a class called ProtocolBuilder in C++. This class should ease the first use of the DataMontitor. It supports the user working with our predefined protocol. Every configuration can be set by functions provided by the ProtocolBuilder. Once familiar with all the settings, the user can easily change the system, the application and the programming language.

4.2. Visualization

The monitoring tool supports different kinds of charts and displays. The data source can configure the desired chart by sending the appropriate control info to the data sink. Visual controls such as analog and digital displays, line charts, bar charts, scatter plots, and pie charts are supported. Instantaneous values are visualized either with the self written analog or digital display. Visualization of the continuous values is accomplished by the use of the free ZedGraph-Graphic library. This library supports a various amount of different chart types. As stated in previous section, the DataMonitor basically supports common chart types, as shown in Fig. 5. Although different applications got different demands on data presentation, the DataMonitor covers a wide range of diagrams. In order to enhance it by a new feature, it has to be integrated in data sink and the data source. Although the monitoring tool was modified, existing applications are not affected by that.

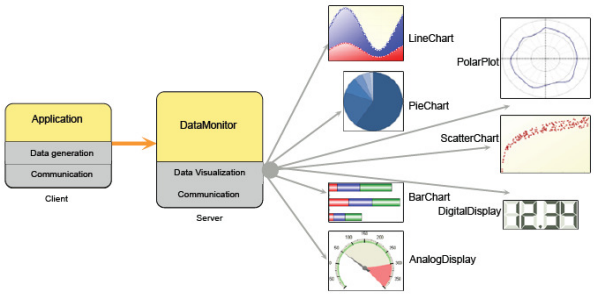

Figure 5: Overview of the system and its supported chart types.

4.3. Implementation

The basic structure of the design is shown in Fig. 6. The design has a clear separation between the communication and the visualization part. Only by that, we can ensure a modular and thus extendable application.

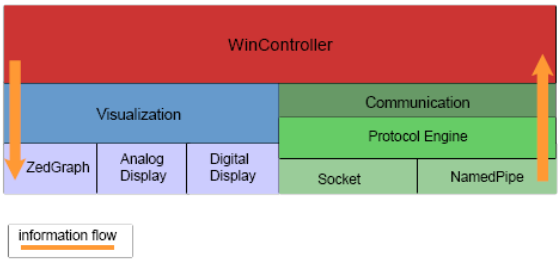

Figure 6: Internal structure and information flow of the DataMonitor.

The core of the software design is the class WinController. It is responsible for controlling the communication and the visualization. Moreover it

contains the state machine shown in Fig. 7.

The state diagram of the defined protocol is as follows:

1. *NoChart*
   Transition from *NoChart* state to *ChartCreated* by NewChart.
2. *ChartCreated*
   Sending measurement data. State remains until CloseChart.

The initial state *NoChart* is reached after the software has been initialized and started up correctly. It is then waiting for incoming requests delivered by the data source. While the server is in the initial state, the only acceptable transition is the creation of a new chart. All other incoming data will be discarded. When creating a new chart, the data source can set the type, the range of the axis, labels, and the amount of input sources, colors, and a few more parameters. While the system is in the *Chartcreated* state, the data source can send chart values, recreate a new chart or close the chart in order to quit or interrupt the measurement phase.

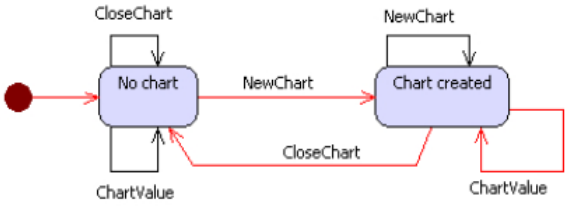

Figure 7: State diagram of the monitoring tool.

The controller acts as a master for the visualization and as a slave for the communication as shown in the information flow in Fig. 6 by the orange arrows.
The communication concept uses the Strategy pattern (Gamma, Helm, Johnson, Vlissides, 1994) in order to select the different interfaces at runtime. Thus it is possible to swap dynamically between the implemented interfaces. After the desired interface was selected and the server started, a thread within the chosen communication object begins running and waits for incoming data. Incoming requests are handed over the protocol engine, which only checks the correctness and the semantic (e.g. keywords, hierarchical order and proper nesting) of the TLV packets. Wrong data is discarded and if the TLV packet was incomplete, the server closes the connection to the client. In every other case, the WinController gets informed about arriving data packets. The new data is processed by the WinController in form of a transition in the internal state machine on the condition of possible transitions. The development of the state machine provides us on the one hand with a clearly structured control sequence and on the other hand makes it is to maintain and extend for future purpose. In order to extend the DataMonitor by a new communication interface, the ICommIF in Fig. 8 has to be implemented.

```
1 public interface ICommIF : IDisposable
2 {
3   event StatusChangedEventHandler StatusChanged;
4   event DataReceivedEventHandler DataReceived;
5   event DataSentEventHandler DataSent;
6   event ErrorEventHandler Error;
7
8   void Start();
9   void Send(string msg);
10  void SendBytes(byte[] buf);
11  void Stop();
12 }
```

Figure 8: Interface definition for communication classes.

The interface consists mainly of two functions to start and stop the communication port. Next to these, there are two more functions for sending data, which are currently obsolete but integrated for future purpose (e.g. extension of bidirectional communication). Incoming data or occurring errors are signaled by three events: StatusChanged, DataReceived and Error. The fourth event DataSent is currently obsolete, because of the unidirectional communication. These signals are either captured by the ProtocolEngine or the WinController. By using the Observer (Gamma, Helm, Johnson, Vlissides, 1994) pattern, we can define this one-to-many dependency between objects so that when one object changes (e.g. new data arrived), all its dependents are notified and updated automatically. Because of this design pattern, the DataMonitor receives data in a non-blocking way.
The visualization component consists of three different libraries: analog display control library, digital display control library, and the ZedGraph library. The analog and digital display controls were developed specifically as dial indicators for speed (analog control) and time (digital control). Nevertheless, the analog control can be used to display pressure values, static state values like ON/OFF and other measurement data (e.g. voltage level). In addition, the digital display control can be used to present textual information as running text (e.g. error messages during a system test, which can be very useful in terms of failures). With the integration of the Open-Source library ZedGraph (ZedGraph, 2008), the functional range could be greatly expanded by many different diagrams. Although it is very powerful, you can create diagrams with a few parameters. This approach fits perfectly into the overall concept of our design – little configuration effort, but a large spectrum of diagrams. For the addition of new charts, the ZedGraph library still offers enough potential. For further details we refer to the online documentation.

5. EXPERIMENTAL RESULTS

In order to provide some results, we intend to give a simple example on the usage of this tool. The example code snippets show, that a few lines suffice to produce well-to-understand visualization of measurement data. Before the client can connect to the server, the desired communication interface has to be selected. After that, the client can start the control sequence to generate a diagram. Fig. 9 shows us the connection routine for a client using socket

communication.

```
Socket mSock = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
    ProtocolType.Tcp);
mSock.Connect("192.168.0.200", 8001); // connect
// start sending data to DataMonitor...
mSock.Close(); // close connection
```

Figure 9: Connect to the server using socket communication.

After the connection to the server was established, the client can send control information to the DataMonitor. In the example shown in Fig. 10, a bar chart with main title, axis title, bar count and elements is created. This information is send to the DataMonitor.

```
BarChart bchart = new BarChart();
bchart.title = "Jahresentwicklung der Rohoelpreise";
bchart.bar_direction = eBarDirection.Vertical;
bchart.x_title = "Jahr";
bchart.y_title = "Euro/Barrel";

bchart.bar_count = 1;
BarElement[] b = new BarElement[bchart.bar_count];
b[0].bar_name = "Euro";
b[0].bar_color = eColor.Green;
bchart.bars = b;

// use ProtocolBuilder
byte[] senddata = ProtocolBuilder.NewChart(bchart.GetBytes());
mSock.Send(senddata, 0, senddata.Length, SocketFlags.None); // send
```

Figure 10: Creation of a bar chart.

When the DataMonitor receives the NewChart control information, the visualization configures the corresponding diagram and the diagram configuration can immediately be check by the user. At this point, the DataMonitor state machine is ready for receiving measurement values. In a few lines, Fig. 11 shows how to generate and send the measurement values to DataMonitor.

```
BarValues bval = new BarValues();

for (int i = 0; i < 8; i++)
{
    // retrieve barchart value from any internal/external source
    ...
    senddata = ProtocolBuilder.ChartValue(bval.GetBytes());
    mSock.Send(senddata, 0, senddata.Length, SocketFlags.None);
}
```

Figure 11: Generation of measurement values.

The result of the presented lines of code is shown in Fig. 12. There you see all labels, the configured coloring and some random bars.

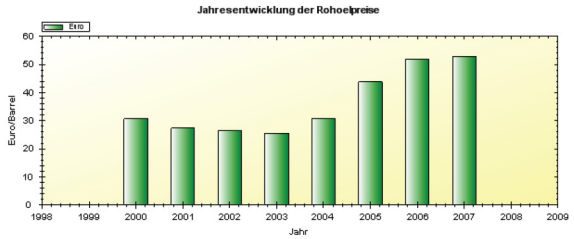

Figure 12: Generation of measurement values.

6. CONCLUSION

The main aim of this work was the development of an easy-to-use, remote and simple to integrate and access (so to speak platform independent) visualization tool. Because of the modular concept of the software design and the user-friendly handling, we assume that the DataMonitor is in addition to the technology field useful in many other areas. Whenever embedded devices, especially restricted devices or so called display-less devices, are used, the need for output types except textual increases. The DataMonitor has been developed exactly for this purpose.

A typical use case for embedded systems in combination with the monitoring tool is to process system information as processor utilization, memory utilization, and power consumption. While running performance tests to identify the maximum throughput of embedded systems, computationally intensive visualization or communication tracers will influence the result. Compared to other common interfaces like Ethernet, the communication effort for data transfer is far less with the simple protocol and will therefore hardly influence the trace of the performance counters.

REFERENCES

BSDZone, 2008. *Berkely Software Distribution*. University of California. Available from: http://www.bsdzone.org [accessed April 2008]

Gamma, E., Helm, R., Johnson, R., Vlissides, J.M., 1994. *Design Patterns. Elements of Reusable Object-Oriented Software.*, 1st ed, Addison-Wesley.

Otto-von-Guericke, 2008. *Grundlagen Visualisierung und Wahrnehmung*. University of Magdeburg. Available from: http://wwwisg.cs.unimagdeburg.de/cv/lehre/VisualAnalytics/material/Bade_Grundlagen-Vis-Wahrnehmung+Infovis.pdf [accessed April 2008]

Colin Ware, 2004. *Information Visualization – Perception for design*, 2nd ed, Kaufman.

Microsoft Corporation, 2005. *Interprocess communications*. Available from: http://msdn.microsoft.com/en-us/library/aa365574(VS.85).aspx [accessed June 2008]

Schwarz, D., 2008. *Mehr Information durch Visualisierung von Daten?* University library Bochum. Available from: http://www.b-i-t-online.de/ [accessed June 2008]

Koch, A., 1993. *Offene Systeme – Interprozesskommunikation in verteilten Systemen*. Springer Verlag.

Barnard, F., 1927. *One picture is worth a thousand words.* Printers' Ink trade journal, pp. 114-115.

Gregory M.N., Hagen, H., Müller, H., 1997. *Scientific Visualization: Overview, Methodologies, and Techniques.* IEEE Computer Society, 1997.

ZedGraph, 2008. *Zegraph – Overview, Samples and Class Documentation* from: http://zedgraph.org/wiki/index.php?title=MainPage [accessed June 2008]