e-Panel (Virtual Control Panel)

Adam Egressy*

Faculty of Electrical Engineering
Czech Technical University in Prague

Abstract

Nowadays, control panels and display boards form an important part of electronic equipment in every branch of the industry. They can be found everywhere, beginning from cars and air planes to factories and huge power plants. All of them have one important goal: to transform machine-measured values into visually sensible form and deliver them to people, who drive a car, pilot an air plane or control a power plant. Our intention is to develop a new application that would become a useful, fully virtual control panel.

Keywords: Control Panel, Display Board, OpenGL, Visualization.

1 Introduction

Our intention is to develop an application that would simulate the behaviour of an electric control panel. Such a device is an equipment that usually consists of measuring instruments, display devices and controlling components. A virtual control panel is a computer program that simulates a real one.

A virtual control panel, compared with a physical one has many advantages. First of them is that it is easy to customize. It is possible to put a new virtual device, remove an old one or change their positions or orders without purchasing any devices or unplugging any cables. It can be done by modifying the program or its configuration. Another advantage is that a virtual control panel is more scalable. If it is necessary to increase the number of components on it, the only thing that may need to be rescaled is the hardware of the computer. There is no need to obtain any new physical measurement devices.

When looking for any available active project developing such an application, we never were successful. The closest project to our intention we found was an Airbus A340 Glass Cockpit Project (see [5]).

In the next sections there are described our experiences with developing our own virtual control panel, called e-Panel. Section 2 deals with the important aspects of a virtual control panel in general, while in Section 3 there are concrete solutions, used in our implementation, explained. The remaining four papers contain testing, conclusions, future works and references related to this paper.

* egresa1@fel.cvut.cz

2 Background

A modern virtual control panel should meet the following requirements:

- It has to be able to read data from various sources simultaneously.
- It has to be configurable by the user.
- It has to be extendible by new features.

Data Sources

A control panel has usually more than one input channels. All the time it is switched on, each component is listening on its input for any new data. New data are usually generated on signal changes and the particular component has to respond to these changes. They usually simply indicate the change on their displays.

We expect the same behaviour from a virtual control panel. In contrast with a physical control panel, a virtual one is just a computer program, which cannot measure any physical quantity. It can only read data from input files and the physical quantities have first to be converted to a string of digital values and then to decimal numbers. That is the responsibility of the hardware.

A possible variety of input files is shown on Figure 1. The sine curve means any analogue input signal and the COM nodes are standard serial, parallel or USB ports of the PC (e-Panel) computer.

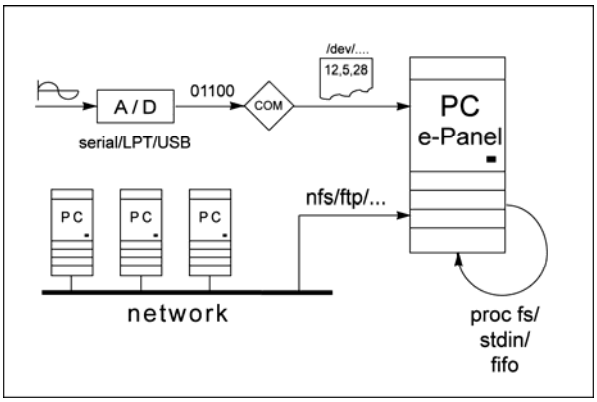

Figure 1: Input signals

In e-Panel, besides the analogue and digital signals, input data can be read from any file on the file system (could it be text file, special device, proc file system, etc.). It has at least two advantages. Fist, it makes possible to visualize computer based data in the virtual control panel (e.g.: outputs from database

engines, values prepared either simultaneously by a separate application or pre-computed). Second, the files do not need to be located on a local data store but they can be read from a network file system (nfs) that is connected to the host machine, as well.

Configurability

The configurability issue is an important property. When developing a virtual control panel, we do not know, how many and what kind of components will be used later, on different installations.

In general, there are two ways how to make an application configurable at user-level. The first way is to attach a configuration file to it. The second way is to develop a configuration interface, which will help users to configure easily the application on their own.

Extendibility

In order to have the application up-to-date, it has to be designed very open for further enhancements. This approach can be reached the best if the application is divided into small modules so that new technologies and new algorithms can get easily integrated in the future.

3 Implementation

This section deals with implementation details of e-Panel. By following its sections, the reader will get a close description about the technologies we used in e-Panel and the reasons why we have chosen to implement them.

Software Architecture

E-Panel is a Linux based application. The Linux operating system was chosen thanks to its open architecture and reliability. While it is usual for Linux based programs, the main programming language of e-Panel is C.

E-panel consists of a number of modules, divided into two different layers (shown on Figure 2). The lower layer contains the application framework, which is responsible for controlling the program flow, reading input data, parsing the main configuration file – generally for all the work that is not concerned about the graphical visualization processes.

Plug-ins, in order to communicate with the framework, have to implement a common interface. More precisely, they have to declare a set of functions, which will be called by the framework later. A more detailed description on plug-ins is given in [1], Chapter 2.

This way we managed to separate the back-end functions from the graphical user interface (GUI) functions in e-Panel, which is an important issue in any plug-in based application.

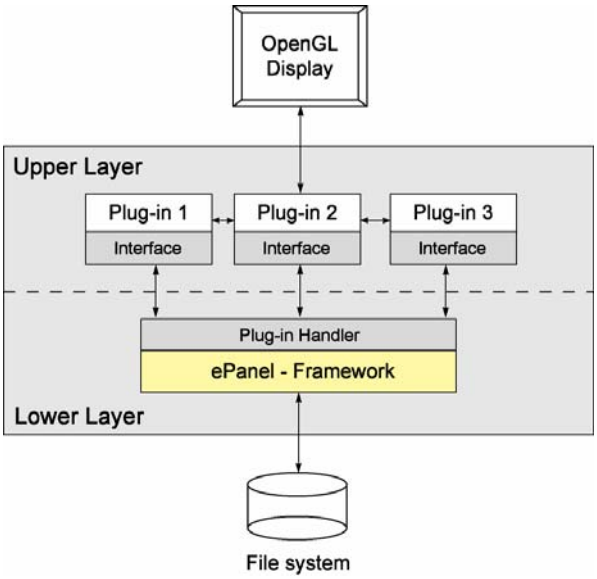

Figure 2: Application architecture

Interaction of Plug-Ins

Plus-ins are allowed to communicate with each other in order to increase their reusability. This capability makes it possible to develop plug-ins that can act as e.g. graphic sub-components used by one or more other controls. In some other cases it could be useful to divide a large plug-in into smaller ones, increasing the modularity of the solution.

A common issue for example concerns text outputs in OpenGL environment. There are a few possibilities how to get text messages printed in an OpenGL scene and it is not a good practice to duplicate all necessary code into each plug-in that wants to print. A better solution is to develop one single library that takes care about the graphical text outputs, load it dynamically on the start-up as one of the plug-ins and use it from your own components.

One can complain about this approach and say that it brings a new kind of dependency into the architecture. This argument is correct because malfunction of a library that is used by one or more other plug-ins will probably cause incorrect behaviour of these components. On the other hand, this risk is compensated with the convenience of modularity we gain thanks to this approach.

Input Data Format

E-panel reads its input data from text files in CSV (Comma Separated Value) format. A file in CSV format is a simple text file containing values separated with comma (rarely with semicolon). These values are often called fields and a group of fields wrapped in one single line is usually named record. In e-Panel we treat each line as a set of separate input sources, which were scanned at the same time.

There are other, more complex formats we could chose but did not. One of the most popular text-based formats intended to use for data exchange is XML (eXtendible Markup Language). The problem with using XML in any virtual control panel is that it is too complicated. The most of the measuring devices we might want to communicate with a virtual control panel are not capable of generating a correct XML output.

Entities of the Application

This section describes the various entities and data structures used in the application. E-Panel uses three main entity types. They are data sources, visual components and plug-ins.

Figure 3 shows the connections between different kinds of entities. Data sources – as discussed in the last section – are files on the file system, where each of them supplies data for one ore more components. Components are visual elements that are placed and rendered on the screen. They acquire information from exactly one data source. Finally, plug-ins are program codes and each of them render one or more components.

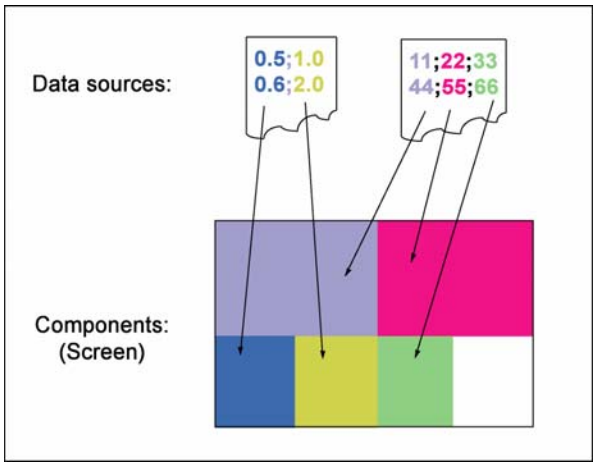

Figure 3: Entities of e-Panel

Configurability of the Application

The configurability of e-Panel is assured by a global configuration file. It has a simple text format that makes it easy to edit from any text editor.

The configuration file begins with information about the size of the application's frame window, which is given by its width and height in pixels. This information should be followed by definitions of all data sources that are to be used. Data sources we define by their identifier (ID) and the particular file name (File) and column number (Column) they come from. See Figure 4 for illustration.

The definition of data sources is followed by the section of plug-ins. A plug-in record is very simple because it contains only two attributes. Figure 5 shows such a record. The first one is the identifier of the plug-in (ID) and the second one is a reference to the file where the plug-in's rendering code is located (File).

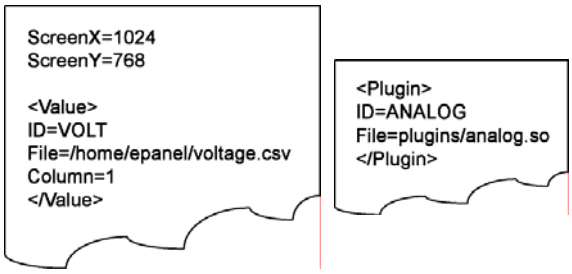

Figure 4: Configuration file (part 1 and 2)

The section of plug-ins should be followed by the largest part, the definition of visual components. Their definitions have to contain required attributes (such as ID, Name, Value, attached plug-in, coordinates and dimensions) and may contains user-defined attributes, which can hold custom information, specific to each plug-in. Please see Figure 6 for illustration.

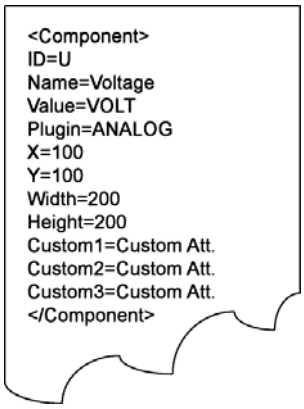

Figure 6: Configuration File (part 3)

Simultaneous Reading of Input Files

As it was discussed in Section 2, input data sources must be scanned for changes simultaneously, because generally it is not possible to predicate, which one of the input sources is going to change next time. A computer program has two possibilities to tap more files at once and check periodically, whether they grow.

The first way, which is used in e-Panel, is to open all the input files in non-blocking mode. Doing so prevents blocking the program flow at run-time if the `read(...)` function is called when no new data is available in a file. More details can be found in [4] Open function. In this case, we can cycle between the input files calling the `read(...)` command in order to determine, whether the specific file did grow since the last call or not.

The second possibility, which is said to be the less efficient, is to create as many threads as many different files we are going to tap. In this case, each thread would issue a call to the `read(...)` function and will be blocked until new data become available. This way was not implemented in e-Panel, because it is much more complicated and less effective because every additional process, thus every new thread, as well, has a negative impact on the operating system's performance.

Rendering Engine and Graphical Subsystems

The rendering engine of e-Panel was developed in OpenGL, which has suitable implementations (either software or hardware accelerated) for the purpose we need. Despite of the fact that the OpenGL library is (by its design) three dimensional, at this time e-Panel simulates a two dimensional virtual control panel. In order to render 2D scenes in OpenGL in a comfortable way, we decided to place the camera right upon the control panel at a fixed altitude and enabled strictly orthogonal projection, which prevents creation of perspectives.

The e-Panel core graphical engine uses the platform independent OpenGL Utility Toolkit (GLUT). This library implements many basic algorithms needed to run an OpenGL based application. For example this library takes care about creating and closing a window. It also implements handlers for various events, such as keyboard input, window got resized, etc. More information can be found in [2] and in [3].

When the e-Panel's screen is refreshed, all the components need to redraw their content. They get invoked by the framework sequentially, one by one, after a painting buffer had been prepared for them. When rendering of all plug-ins has finished, the framework draws the content of its buffer to the screen. The following pseudo-codes illustrate the procedure.

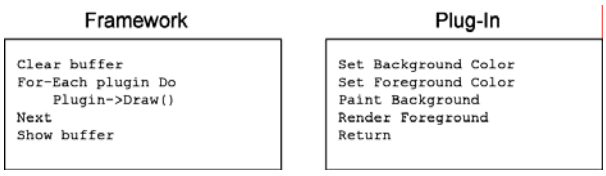

Figure 7: Rendering procedure

Overlapping of Components

Components on a control panel can overlap and for this case it is necessary to calculate with visibility issues. Although this problem may seem insignificant, we can find many instances of it even on seemingly simple and ordinary panels.

One of the simplest examples could be a LED indicator placed on a pointing instrument. A LED indicator usually has small proportions and it makes it easy to place on another control. There can be found an example of this case on Figure 9, where a green light-indicator is placed upon the engine speed-indicator.

There are two possibilities how to solve the problem when one control is partly or entirely covering another one. First, if either the application framework or the plug-ins themselves are clever enough, there can accept a greyscale texture and treat it as its alpha channel for rendering. The alpha channel is numeric information describing the opacity of an object or a texture. This way, controls can be made partly transparent and in this case overlapping is not senseless anymore.

The second solution for the overlapping problem consists in organizing controls into separate layers, where all objects in one layer are placed into a constant altitude. In this case, we can configure each component by specifying its altitude (or the altitude of its parts) with an additional parameter. For example, the problem of the light-indicator on Figure 9 (described in the beginning of this section) can be simply solved by putting the background of the engine speed-meter below the green light-indicator and situating both of them below the pointer of the engine speed-meter. This organization is probably very similar to the placement of such components on a real dashboard and can be achieved in e-Panel by adding three new attributes for these components into the global configuration file, specifying the altitudes of the background, the led indicator and the pointer.

Examples

This section contains figures coming form real screenshots, illustrated by guidelines and commented briefly.

Figure 8 illustrates a user-defined control panel layout. It contains eight components, each of them based on the same plug-in (which just displays the component's name aligned to its centre). The top left control, which is coloured light red, shows the meanings of coordinates and dimensions of components.

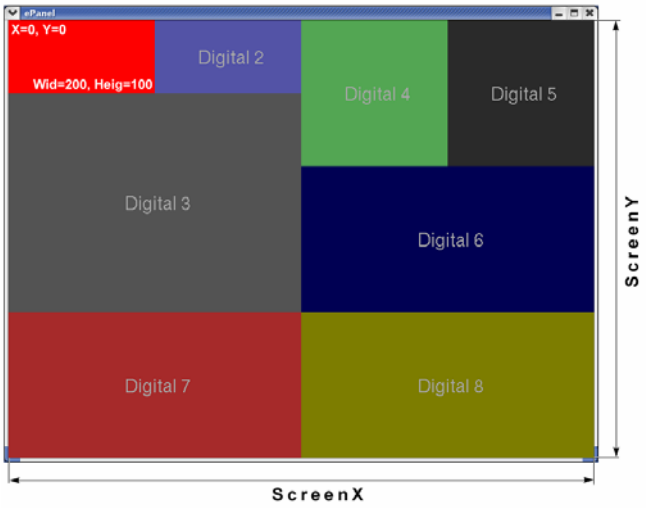

Figure 8: A user-defined layout

Figure 9 shows a real demo scenario of e-Panel, which consists of pointing instrument components and digital display components, backed with a texture of a dashboard in a car.

On Figure 10, the reader can see how the components are arranged on the dashboard. An interesting note is that for example components 1 and 5 or 2 and 10 cover each other. However, it is not a problem until we can uniquely define, what order should the components rendered in.

Figure 9: A dashboard in a car

Figure 10: Parts of a dashboard in a car

4 Testing

Testing of e-Panel was made according to three testing scenarios. All of them are performance test that we designed to gain information about scalability of e-Panel with number of different loads. All of the test cases ran on the following hardware: PIII 700 MHz, RedHat Linux 9 installed in a VMWare Workstation 4.5.2 on hosting operating system Windows XP SP2. The graphics adapter was Trident CyberBlade-Ai1 (AGP) with software emulated OpenGL (Mesa version 6.2).

The first testing scenario consists of five separate tests of e-Panel in action. We tested drawing of 1 to 200 controls in each cycle. In order to collect relevant information about the performance parameters of the whole e-Panel framework, not only of the concrete control plug-in in the particular tests, we decided to use the simplest control developed, the digital instrument. (See Figure 10, part 7, 8 or 9 for example.) The results of the first series of tests can be seen on Figure 11. It shows us that on a very basic hardware that the first slight increase of time needed to refresh the control panel's desktop occurs when redrawing 50 to 100 components simultaneously.

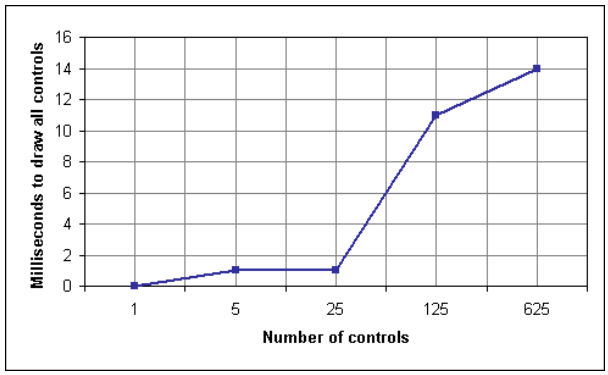

Figure 11: Performance test 1 – number of controls

The second scenario, which can be called dependency of overall performance on number of columns of input files, pleasantly surprised us. We've tested a configuration with one input file, having from 1 to 500 columns and didn't find any scalability bottleneck. Even in with 500 columns in one single CSV file, the time spent with reading and parsing all the columns did not exceed 1 millisecond. Please see Figure 12 for details.

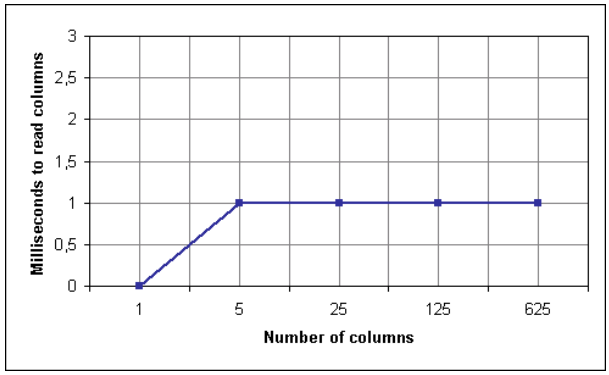

Figure 12: Performance test 1 – number of columns of input files

The aim of the last series of tests was to study the dependency of overall performance of the application on the number of separate input files used in a configuration. There were five different configurations tested and the results we found surprisingly satisfying. The results are included in the graph on Figure 13. It shows us, that no matter, whether there is 1 or there are 125 input files, we don't waste more than 1 millisecond when reading all of them. The only problem that could occur consists in the fact that operating systems allow a limited number of files (more precisely file descriptors) to be open at once. On the other hand, this number is usually larger than 1000 and that's why it doesn't seem to be a real scalability problem.

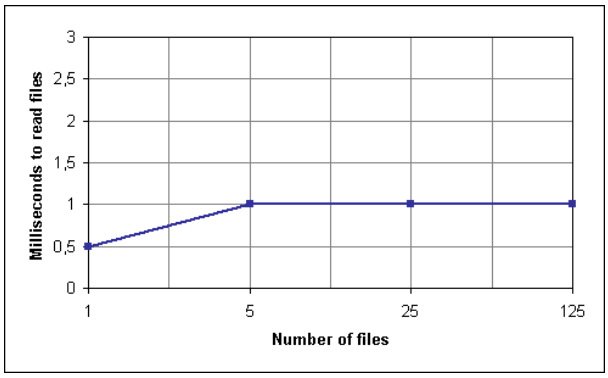

Figure 13: Performance test 1 – number of input files

5 Conclusions

The objectives of our projects were completed - we have developed a new virtual control panel application. The framework of our application meets all the three requirements, we had set up in Section 2:

- It can process multiple data sources simultaneously.
- It is configurable on user-level.
- It is extendible with new features.

There are also three different graphical plug-ins that we have developed within this project. They are the following:

- Digital measuring instrument control.
- Analogue pointing instrument control.
- Graph control.

Although the current version of e-Panel is a final release that seems to be very stable, there are already ideas how to improve it in a future work. Please see Section 6 for details.

6 Future work

Future work on e-Panel should focus with high priority on interactivity and new graphical plug-ins. Further future work could aim on support for input filter plug-ins and a multiplatform version.

At this moment, there is a lack of interactivity support in e-Panel. The application is able to simulate various controls on a display board that brings visual information to the user, however it is not able to handle inputs from the user. Once interactivity support is added, it will greatly enhance the usefulness and the overall benefit brought by this application.

New visual plug-ins do not only bring new look for the current components, but also extends the range of places, where e-Panel becomes useful. For example, a base component for 2D matrix displays would make us possible to create controls like GPS navigation, smart oscilloscopes, custom matrix displays, etc.

A multiplatform build of e-Panel would increase the number of computers, where the program can run. This way, we would not eliminate users, who are not familiar with Linux operating systems.

Support for input filter plug-ins, however is not the highest priority question for us, would be able to manipulate with input signal right before they arrive at the corresponding plug-ins.

References

[1] M. Mitchel, A. Samuel: *Advanced Linux Programming*, 2001, New Riders Publishing, ISBN: 0735710430

[2] OpenGL Architecture Review Board. *OpenGL Programming Guide – The Redbook*. Addison-Wesley, 2003, ISBN: 0321173481

[3] M. J. Kilgard. *OpenGL Programming for the X Windows System*. Addison-Wesley, 1996, ISBN: 0201483599

[4] Linux Manual Pages

[5] Airbus A340 Glass Cockpit Project (http://a340gc.iradis.org/about/index.en.html)