Human Machine Interface for Piezo Control System

Paweł Plewiński, Dariusz Makowski, Aleksander Mielczarek, and Andrzej Napieralski

***Abstract*—The increasing complexity of scientific experiments puts a large number of requirements on the systems, which need to control and monitor the hardware components required for the experiments. This article discusses three different technologies of developing Human Machine Interface (HMI) for the Piezo Control System (PCS) developed at the Lodz University of Technology (TUL). The purpose of the PCS system is compensating the detuning of a superconducting accelerating structure caused by the Lorentz force. A set of full-custom HMI operator interface was needed to operate the prototype device. In order to select the technology, which would best suit the requirements; the interface was designed and developed in native Qt, Qt Quick and EPICS. The comparison was made and their advantages and drawbacks are presented.**

***Index Terms*—Human Machine Interface, HMI, High Energy Physics, Qt, Qt Quick, QML, EPICS, BOY, MicroTCA**

I. Introduction

THE increasing complexity of scientific experiments, such as particle accelerators, free-electron lasers or tokamaks, puts an ever-growing set of requirements on the systems, which control and monitor the hardware set-ups [1, 4]. One of the crucial requirements is to provide reliable and ergonomic user control panels for operation, management and state monitoring of the aforementioned systems.

This article presents and compares three different technologies of creating Human Machine Interface (HMI) for physical devices. Three various attempts are discussed, namely native Qt user interface, Qt-QML with Qt Quick 2 interface controls and Best OPI Yet (BOY).

II. Controlled System

The motivation for developing the discussed HMI is the urge to control and monitor a prototype Piezo Compensation System (PCS) developed by the Department of Microelectronics and Computer Science of the Lodz University of Technology (TUL-DMCS) for the European Spallation Source (ESS) within the in-kind contribution of Polish Electronic Group (PEG) consortium.

The PCS system structure is presented in Figure 1. Its electronic part consists of a MicroTCA.4 chassis with a number of modules, namely: Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs) in a form of FPGA Mezzanine Carrier (FMC) modules hosted on an FMC carrier, prototype high power dual-channel amplifier assembly (shown in Figure 2). Optionally a strain readout device using an S-type full-bridge tensometer can be connected. The modular approach makes it possible to build and evaluate the complete system more easily and with lesser cost, as well as to easily test several possible components without requiring significant changes in the system.

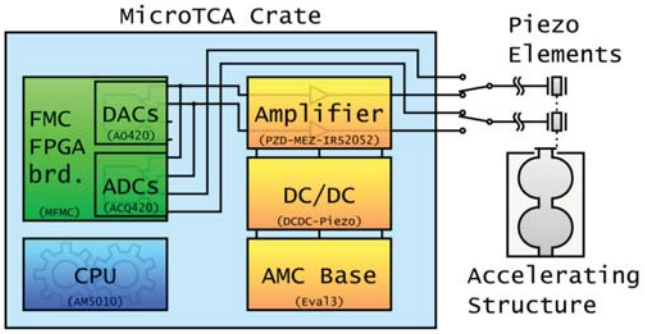

Figure 1. Block diagram of Piezo Compensation System prototype

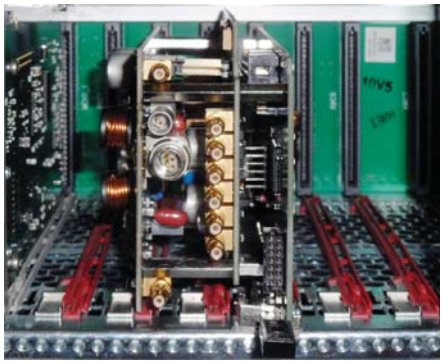

Figure 2. MicroTCA.4 piezo driver prototype

The system interfaces two piezo elements; each can be used either as a sensor or as an actuator. The purpose of the PCS system is compensating the detuning of a superconducting accelerating structure caused by the mechanical stress. The detuning is mainly caused by the deformation due to Lorenz force due to high electric field gradients inside the cavity and also in some part by the microphoning effect. First factor can be compensated by applying a predefined force pattern to the structure, synchronously to the accelerator timing. The second one requires observation and the active correction of the cavity deformation. Therefore, in a final system one piezo element will be operated as an actuator, whereas the second one will be used as a sensor.

P. Plewiński, D. Makowski, A. Mielczarek and A. Napieralski are with the Department of Microelectronics and Computer Science, Lodz University of Technology, Łódź, Poland (email: pplewinski@dmcs.pl).

To test the developed system, firstly, it was required to prove that the signal from piezo sensor can be reliably read with the prepared hardware. Secondly, it had to be shown that the piezo actuator can be properly driven and effectively shift the resonant frequency of the accelerating cavity. In order to do so the system shall offer the functionality of an oscilloscope and an arbitrary function generator synchronized to the machine timing. To fulfill this requirement a 4-channel generator and oscilloscope was developed in an Artix 200 FPGA circuit of the MicroTCA.4 FMC carrier module [2]. It interfaces with the external world by means of two commercial FMC modules containing the ADC and DAC circuits. The device is configured and controlled over a PCIe interface, exposing a simple set of registers accompanied by memory regions containing the received and generated waveforms.

The aforementioned hardware is connected to an x86-based embedded industrial computer installed in the MicroTCA.4 chassis [3]. The FMC carrier card is connected to the computer via PCI Express link on the backplane. The whole FMC assembly is visible as a single device. A custom driver developed at TUL-DMCS is used to access the device.

The system operation with piezo actuators was initially tested with use of the load cell. In this test set-up the commercial load cell readout device is connected to the PC with an USB-to-RS485 adapter and is visible to the software as a virtual serial port. The meter implements a custom binary protocol, sending a constant start byte followed by the strain value measured by the tensometer every 80 ms in fixed-point decimal format (4 bytes). The practical tests were done in the FREIA facility in Uppsala, Sweden, with use of a spoke cavity operating in the self-excited loop mode with the electric field gradients of around 1 MV/m.

III. Requirements

User interface was needed to operate the prototype device. The HMI provide readouts from tensometer and ADCs updated in real-time as well as allow controlling the waveform generator implemented for the DACs. These allow independent control of all four channels; provide possibility of selecting waveform generation from a pre-defined list and adjustment of signal parameters, such as amplitude, frequency and offset.

Ergonomics and readability are among the most important qualities of good HMI [1, 5, 6, 7]. Moreover, the underlying technology should offer performance allowing use of the interface even on embedded computers with limited resources and computing power.

A significant advantage of the selected HMI design technology should be the ease of modifications of the user interface for the needs of specific users. An example would be a possibility of easy development of an interface with limited functionality for less advanced users of a system. The platform should provide tools for easy development and modification of user interfaces, preferably in a WYSIWYG (What You See Is What You Get) fashion.

Also a functionality of remote and distributed access to the controlled systems, preferably from multiple clients, would be advantageous.

IV. Qt Framework

Qt is a C++ based cross-platform software programming framework developed and supported by The Qt Company [7]. The current stable major version of the framework is 5, first released in 2012 and still actively developed. The framework provides not only graphical user interface, but also modules supporting more advanced functionalities such as multimedia, access to databases or computer peripherals, such as serial port. Most of these modules are available for all platforms supported by Qt [8].

Unique features implemented in the framework are the system of signals and slots and property system. The former is an event system, which allows binding individual components, while the latter provides possibility to create sophisticated component parameters, with certain actions executed on change of the value [9, 10].

The Qt framework provides two technologies for creating applications equipped with GUI – traditional native user interface and newer Qt Quick and both are discussed in sections below. Applications supporting the PCS system were developed in both technologies for comparison.

V. Native Qt

Native programs using the Qt framework can be built for Windows, Linux and Mac OS, among others. The application code and user interface code are written in C++. The interface layout can also be placed in an XML file, which is converted to a C++ header file during the compilation process [11]. The applications using Qt framework follow the user interface style of the host operating system and therefore integrate seamlessly with the particular graphical environment.

The HMI developed in this technology were divided in two applications – first for displaying readouts from tensometer and the other for controlling signal generators and displaying signals gathered on the ADCs in the time base. This division follows the physical structure of the system.

In the tensometer application, the main part of the interface is a chart view of the readout. Available UI controls include: selection of the port, to which the measurement device is connected, setting tare tension and control of chart scaling. Screenshot of the application is shown in Figure 3.

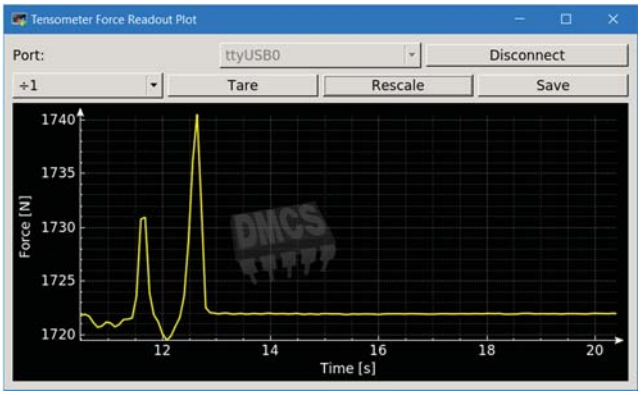

Figure 3. Load cell readout application developed in native Qt

The other application has much more complex interface, due to the fact that it needs to present more data and provide control over greater number of parameters. Again, the major part of the window is occupied by a chart view and chart-related controls and the other part of the window is used for controlling parameters of the generator and signal acquisition. Screenshot of the main window with UI for signal generation and acquisition are presented in Figure 4.

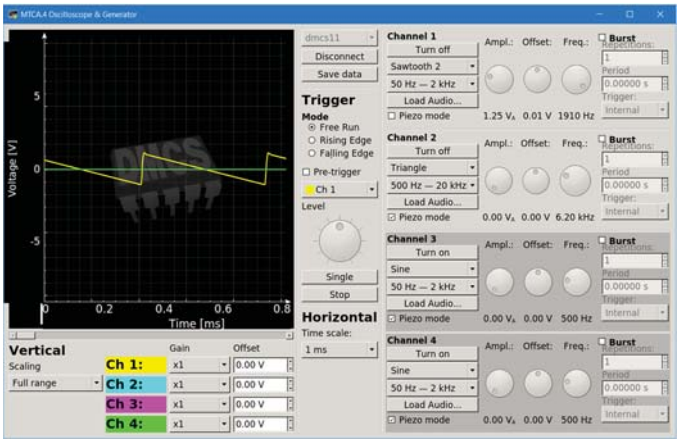

Figure 4. Main PCS application written in native Qt for controlling signal acquisition and generation.

Since plotting charts was one of the key tasks of the discussed applications, an efficient plotting subsystem was required. While Qt integrates a plotting module called Qt Charts, its licensing terms made it impossible to be used in the discussed applications. Therefore, the decision was taken to utilize chart widget from an open-source QCustomPlot library [12]. The library provides a wide range of customizable charts and is available under attractive LGPL license.

Since the native Qt 5 platform is mature and partially backwards compatible with its older version – Qt 4, the choice of stock widgets is reasonably large, however they often need several modifications in order to be intuitively used in an operator user interface. The performance of the native code is very satisfying and achieving refresh rates of over 10 Hz on an embedded Kontron AM5010 PC in MicroTCA chassis was not a major issue [3].

Due to the fact that the developed HMI runs as a single native application, adding functions interacting with local computer, such as exporting logged data to file or loading arbitrary waveform to FPGA memory was not a problem.

The framework is packaged with a Qt Creator Integrated Development Environment (IDE), which integrates a powerful source code editor and an intuitive graphical UI editor, and covers all steps of application development.

On the other hand, the code responsible for UI is directly bound with the application logic. Programming model suggested for this technology does not promote separation between the UI layer (View) and application logic. Moreover, since the interface is implemented in C++ code, any change of layout requires recompiling the whole application and often also doing changes in the program code.

Unfortunately, the Qt framework does not provide high level libraries for network discovery and message-based communication, therefore additional work would be required to transform the discussed HMI applications into a distributed system.

VI. Qt Quick

Qt Quick is a relatively new technology available in the Qt Framework. It introduces a new language for interface description – QML, and the interface code can be enhanced using Java Script. Applications using this technology can be developed not only for all the platforms supported by native Qt code, but also for mobile platforms as Android, iOS or Windows Mobile [13].

All the HMI solutions implemented in this technology are contained in a single application, divided into tabs, which represent the subsequent components of the system. In this approach an additional tab for limits configuration was also provided. Screenshots of individual operator interfaces are shown in Figure 5, Figure 6 and Figure 7.

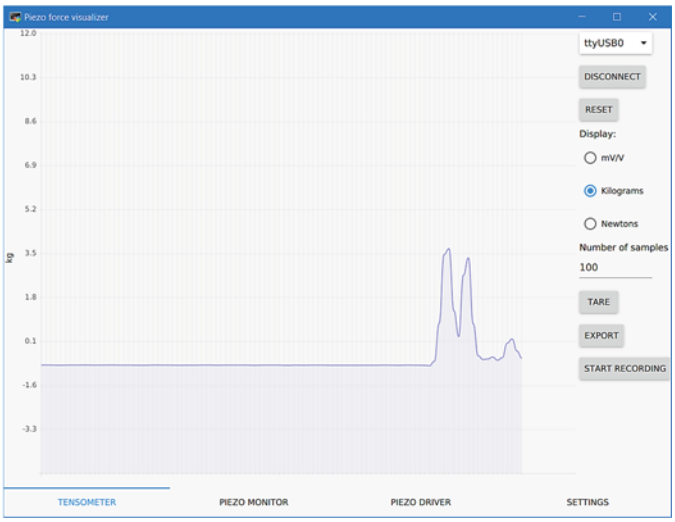

Figure 5. Tensometer readout UI in the Qt Quick application.

Figure 6. Panel for configuration of signal generation parameters in the Qt Quick application.

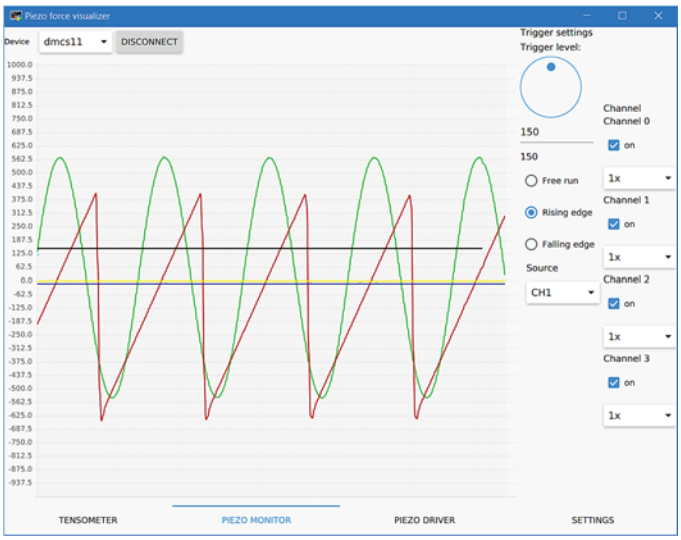

Figure 7. Signal acquisition interface developed in Qt Quick.

The UI code can access appropriately annotated C++ code, i.e. annotated methods of objects inheriting from QObject [13]. Thanks to this, the code for communication with the devices written for the previous application could be reused with only minimal changes. This also means that a Qt Quick application can have exactly the same functionality as a native one.

The UI is adjusted for touch screens. Complex support for touch gestures is provided by the environment and gestures are supported by the default widgets. Therefore, an operator could intuitively use such interface on a mobile device such as tablet, e.g. during on-site works.

The QML is a declarative language and as such it minimizes the number of code required to implement desired functionality. Thanks to the advanced property system, bindings between different properties of components only need to be specified and not explicitly implemented. The programming model requires dividing the application into reusable components. Moreover, the interactions between respective UI elements and interactions between UI and application logic are described only on higher level. Therefore, it is easy to customize the user interface without the need to modify code and it does not require recompiling the application.

On the other hand, the technology has also significant drawbacks which caused the authors to stop the development using this technology. Firstly, the choice of available widgets is currently very limited, most likely due to relatively low adoption of this technology. The widgets provided with the framework were scarce and their behavior did not always meet the requirements.

QChart.js chart library was used in the project [13]. It was the only plotting library for Qt Quick technology available on a permissive LGPL license. It is a port of Java Script Chart.js library to the QML platform. Its major flaw is low performance due to the technology used. Stuttering was observable even at refresh rates of 5 Hz with a single plot and attempts to increase either the refresh rate or number of plots made the whole application barely usable.

Similarly to the native Qt application, any network functionalities would have to be implemented from scratch or with the use of external libraries or message brokers.

VII. EPICS

A. *EPICS environment*

Experimental Physics and Industrial Control System (EPICS) is an open-source set of tools, libraries and applications aimed at creating distributed real-time control systems for scientific applications [15]. It provides a standardized Channel Access (CA) protocol, which is used by all EPICS-compliant tools. It is designed to provide low latency and high bandwidth and can be reliably used even in large networks. EPICS is widely used in scientific experiments around the world [16, 17].

Client/Server and Publish/Subscribe techniques are used by EPICS for communication over network effectively, i.e. without overhead for sending unneeded data [15]. Servers in the EPICS ecosystem are called Input/Output Controllers (IOC) and provide information accessible over CA protocol in form of Process Variables (PV). They usually contain code for controlling physical devices.

Several variable types exist in EPICS. The most commonly used are analog input and output (ai, ao), which represent floating point values, long input and output (longin, longout), binary input and output (bi, bo), multi bit binary input and output (mbbi, mbbo) and calculation variables (calc). They can include additional parameters, which characterize the variable purpose and functioning. Examples of these parameters are description, unit, names and values of respective binary values, alarm configuration or scanning rate.

Each variable is defined in a specific record in database file. A record contains variable type and name and can contain additional parameters. An example of variable definition record is presented in Listing 1. It presents an output field for configuration of signal acquisition trigger parameters with named predefined values.

```
record(mbbo, "PIEZO:OSC-CH$(CHANNEL)-TRIG")
{
  field(DESC, "Signal trigger")
  field(SCAN, "1 second")
  field(PINI,  "0")
  field(DTYP, "asynInt32")
  field(OUT,  "@asyn($(PORT),$(ADDR),$(TIMEOUT))OSC_TRIG")
  field(ZRVL, "0")
  field(ONVL, "1")
  field(TWVL, "2")
  field(THVL, "3")
  field(FRVL, "4")
  field(ZRST, "None")
  field(ONST, "Free")
  field(TWST, "Rising edge")
  field(THST, "Falling edge")
  field(THST, "Any edge")
}
```

Listing 1. Example record definition in EPICS database

B. *BOY operator interface environment*

There are multiple ways to receive data from a CA server, not only by means of console utilities or libraries for several programming languages, but also by intuitive operator interface.

Several solutions for designing EPICS-compatible HMI exist and the most widely used is BOY (Best OPI Yet) [18].

BOY provides development environment based on open-source Eclipse IDE, called Control System Studio (CSS) [18]. It is both editor and runtime environment for the HMI developed with this solution.

The interface designs are defined in OPI files. Each OPI file can contain widgets or other embedded OPI files. Reuse of interface design is possible by creating template files with macros, which are substituted with requested values.

BOY provides an abundance of widgets well suited for scientific applications. All widgets can monitor an EPICS PV or a local variable. Thanks to the subscribe mechanism of EPICS, the values displayed for the user are refreshed only when the value has actually changed, which saves computer and network resources. If the functionality offered by the standard widgets does not fulfill the requirements, new widgets can be developed and programmed in Java.

C. Device support

To make it possible to control and monitor a given device using EPICS, so called device support must be present. While there exist device support packages for multiple devices available on the market, the custom FPGA based data acquisition and signal generator obviously did not have one. Similarly, the commercial tensometer data reader was not provided with EPICS support. Thus, it was necessary to provide it for both devices.

Generally, device support for EPICS can be written in C or C++. However, the API provided by EPICS is not always intuitive and requires substantial work to handle asynchronous requests. Therefore, AsynDriver was developed, which can be used for either asynchronous or synchronous device support [19]. Moreover, to simplify the process of writing drivers for devices, which use ASCII-based protocol (over serial port, GPIB or web sockets), StreamDevice library exists, which can automatically handle even complicated text protocols [20].

The C++ version of AsynDriver was used to provide support for both devices used in the system described in this paper. StreamDevice was evaluated for use with the tensometer, but it had problems handling the binary protocol of the device.

For both devices, a major part of the code responsible for the communication with hardware could be ported from the existing Qt solution with only small changes.

D. Human Machine Interfaces

Operator UI was developed for all the supported devices. Three main UIs were developed, i.e. for ADCs, signal generator and tensometer. These panels are separate, but thanks to the functionality of BOY, embedding them all in a single user interface would not be complicated. Similar functionality as in the other tools mentioned in this paper is provided. Screenshots are visible in Figure 8, Figure 9 and Figure 10.

Thanks to the templating system and possibility of embedding panels in one another it is possible to easily develop a more complex interface, which would integrate several other HMIs. Moreover, they can be developed and edited by users without programming capabilities, since the included widgets

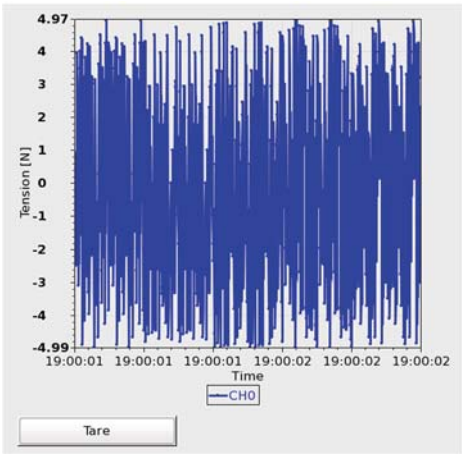

Figure 8. Tensometer readout UI developed in BOY (EPICS).

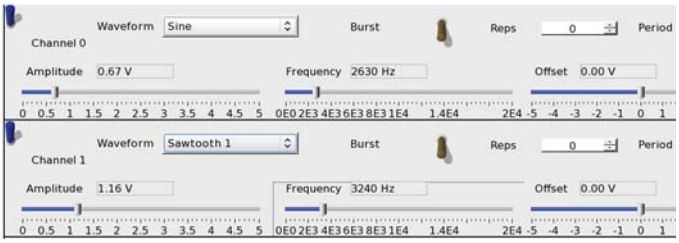

Figure 9. User interface for configuration of signal generation parameters developed in BOY (EPICS).

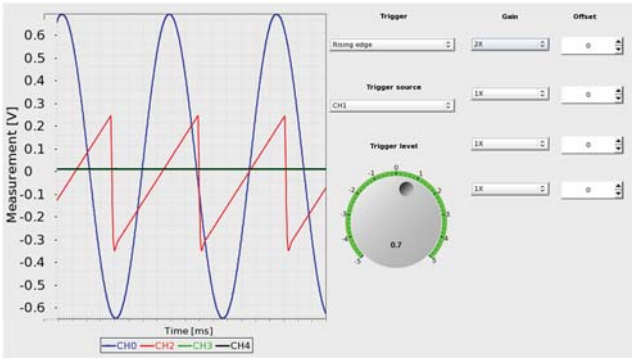

Figure 10. Signal acquisition interface developed in BOY (EPICS).

can handle the PV protocol themselves. BOY provides good separation between UI and controller code, as the whole communication is realized by the EPICS PV layer.

VIII. Conclusions

EPICS is the state of the art control system used in numerous high energy physics experiments and other control systems worldwide. The client-server architecture with standardized and efficient protocol makes it easier to develop functional control systems. Moreover, the EPICS ecosystem consists of several tools for building operator user interfaces, which ease up the development and modification of GUIs fully compliant with EPICS CA protocol. Support for many common devices for EPICS has already been developed and is available on permissive licenses, which can further simplify the development of complex control systems.

On the other hand, the strictly defined structure of EPICS makes it difficult to implement certain non-standard functionalities. For example, support of generation of arbitrary

waveforms would require major work both in IOC and in BOY, e.g. with the use of scripting. Thus, native applications can be a good choice for maintenance or testing applications, which would need to exchange large amounts of binary data and for which remote and distributed control is not a priority.

Qt Quick is a promising technology; however it is not yet ready for use in control system applications due to the poor performance on embedded hardware with low processing power and without dedicated graphics processor. Little availability of quality widgets is also an issue. Yet, in the future, after the increase of computing power in embedded systems, it may be a good successor of the native Qt user interface. Moreover, as the QML interface description language is extendable, it is possible to implement an EPICS client with Qt Quick interface defined in this language, which may be a successor of the current Java-based tools. This would give an additional benefit of the possibility to run such a client on mobile devices, which can be useful e.g. for onsite works.

REFERENCES

[1] Arun K. Ghosh, “Introduction to Measurements and Instrumentation”, 4th edition, October 2012, PHI

[2] Advanced Industrial Electronic Systems, “MFMC MTCA.4 FMC Carrier Module Overview”, 2014

[3] Kontron, “AM5010 User Guide”, Rev. 4.0, May 2011

[4] D. Makowski, A. Mielczarek, P. Perek, G. Jabłoński, A. Napieralski, M. Orlikowski, B. Sakowicz, P. Makijarvi, S. Simrock and V. Martin, 2014 "High-Performance Image Acquisition and Processing System with MicroTCA.4," 19th IEEE-NPSS Real Time Conference (RT), pp. 1–2, May.

[5] P. Asensio, P., R. Vilanova, and B. Amante García. "Human intervention and interface design in automation systems." International Journal of Computers, Communications & Control 6.1 (2011): pp. 166-174.

[6] Deborah J. Mayhew, “Principles and Guidelines in Software User Interface Design”, Prentice Hall PTR, 1992, pp. 579-596

[7] John J. Pannone, “How to best design an HMI system”, Product Design and Development, June 2015

[8] The Qt Company, “The Qt Company”, https://www.qt.io/company/, accessed 1.05.2017

[9] The Qt Company, “Qt 5.9 Documentation: Qt Overviews”, http://doc.qt.io/qt-5/overviews-main.html, accessed 1.05.2017

[10] The Qt Company, “Qt 5.9 Documentation: The Property System”, http://doc.qt.io/qt-5/properties.html, accessed 1.05.2017

[11] The Qt Company, “Qt 5.9 Documentation: Signals & Slots”, http://doc.qt.io/qt-5/signalsandslots.html, accessed 1.05.2017

[12] The Qt Company, “Qt 5.9 Documentation: User Interface Compiler (uic)”, http://doc.qt.io/qt-5/uic.html, accessed 1.05.2017

[13] Emanuel Eichhammer, “Qt Plotting Widget QCustomPlot”, http://www.qcustomplot.com/index.php, accessed 4.05.2017

[14] The Qt Company, “Qt 5.9 Documentation: QML Applications”, http://doc.qt.io/qt-5/qmlapplications.html, accessed 1.05.2017

[15] Jutting Bytes, “QChart.js: QML Bindings for Chart.js”, February 2014, http://jwintz.me/blog/2014/02/15/qchart-dot-js-qml-binding-for-chart-dot-js/

[16] EPICS, “About EPICS”, http://www.aps.anl.gov/epics/about.php, accessed 10.05.2017

[17] J. Zelaya, D. Makowski, P. Perek and A. Napieralski, "Human Machine Interface for data acquisition systems applied in High Energy Physics," 2016 MIXDES - 23rd International Conference Mixed Design of Integrated Circuits and Systems, Lodz, 2016, pp. 93-98.

[18] T. Katoh et al., 2005, "Present Status of the J-PARC Control System," Proceedings of the 2005 Particle Accelerator Conference, pp. 302–304

[19] Control System Studio, “Control System Studio”, http://controlsystemstudio.org/, accessed 10.05.2017

[20] Mark Rivers, “asynDriver: Asynchronous Driver Support”, http://www.aps.anl.gov/epics/modules/soft/asyn/, accessed 10.05.2017

[21] Dirk Zimoch, “EPICS StreamDevice”, http://epics.web.psi.ch/software/streamdevice/doc/, accessed: 15.05.2017

Pawel Plewiński received MSc degree in computer science from the Lodz University of Technology (TUL), Poland in 2016. His main areas of interest are embedded and mobile systems and software development. He has been involved in projects coordinated by such facilities as Deutsches Elektronen-Synchrotron (DESY), European Spallation Source (ESS) and International Thermonuclear Experimental Reactor (ITER).

Dariusz R. Makowski received the M.Sc. degree and the Ph.D. degree (with honours) in 2001 and 2006, respectively, and a D.Sc. degree in electronic engineering in 2018 from the Lodz University of Technology, Poland. Since 2001, he has been with the Department of Microelectronics and Computer Science (DMCS) at the same university. He is currently the Professor Assistant at the Lodz University of Technology.

He is focusing his research on digital control, data and image acquisition or processing. Dariusz Makowski is leading the Control and Data Acquisition (CADAQ) group involved in development of complex, distributed controls systems based on Advanced Telecommunications Computing Architecture, MicroTelecommunications Computing Architecture and PXI/PXIe standards. The developments cover the whole path from the hardware design, drivers, low and high-level firmware programming including HMI.

He has authored or coauthored 3 books and over 160 papers published in conference proceedings and journals. Since 2001 he collaborates with DESY in Germany, since 2008 with ITER in France research institutes. He has received 15 international and 22 national awards including: Prize of the Prime Minister of Poland in 2007, Lodz University Dean's Awards, Minister Scholarships, Award for Young and Talented Scientists. He gave 6 invited talks and seminars in USA and 5 European countries. He has been a member of MIXDES international conference program committees.

Aleksander Mielczarek received his Ph.D. degree in electronics at the Faculty of Electrical, Electronic, Computer and Control Engineering in 2017. His main areas of interests are high-speed data acquisition and processing, gigabit communication, integrated sensors and embedded systems. He is involved in the development of control and data acquisition systems for European X-Ray Free-Electron Laser (E-XFEL), a project carried by Deutsches Elektronen-Synchrotron (DESY). He also participates in the realization of Piezo Compensation System for European Spallation Source (ESS).

Andrzej Napieralski received the MSc and PhD degrees from the Lodz University of Technology (TUL), Poland in 1973 and 1977, respectively, and a DSc degree in Electronics from the Warsaw University of Technology, Poland and in Microelectronics from the Université de Paul Sabatiér, France in 1989. Since 1996 he has been Director of the Department of Microelectronics and Computer Science. Between 2002 and 2008 he was Vice-President of TUL. He is an author or co-author of over 960 publications and editor of 19 conference proceedings and 12 scientific Journals. He has supervised 45 PhD theses; six of them received the Prime Minister of Poland prize. In 2008 he received the Degree of Honorary Doctor of Yaroslaw the Wise Novgorod State University, Russia.