electrocardiography, ECG recorder,
QRS detection, digital signal processing,
Python language, Linux, open source

Wojciech KOSIŃSKI*, Aleksander OWCZAREK*,
Bartłomiej JAROCKI*, Michał MOMOT*, Marcin PŁACZEK*,
Grzegorz ZEGARTOWSKI*, Adam GACEK*

DIGITAL SIGNAL PROCESSING IN ECG RECORDER WITH PYTHON-BASED SOFTWARE

The aim of the paper is to present the possibilities and the advantages of using open source solution like Python and Linux in medical application development. An implementation of the QRS detection and classification is described as an example of integration of C++ and DSP toolkit in a Python application.

1. INTRODUCTION

Nowadays information technology world demands efficient methodologies of software engineering resulting in fast time-to-market production cycles. One of the important parts within such methodologies is selection of appropriate tools, including programming languages in the first place. Medical information technologies put stress on another requirement – reliability. Recent years showed very fast and effective development of the so-called free or open-source software solutions (mostly under GNU Public License) with its prime representative – Linux operating system.

Open-source systems have proven to be a stable and reliable platform for all sorts of task ranging from server, through desktop, ending in miniature even handheld embedded applications. These systems give medical software developers a lot of freedom, and most of all, possibility to adjust available components to there, sometimes very specific requirements, and easily migrate among available hardware platforms and operating systems.

Thus reliability, portability and flexibility made open source software our solution of choice for medical device software development. Efficiency objectives, both in aspects of development time and computational power, drew our attention to combination of Python and C++ programming languages, which resulted in successful working applications for portable devices as well as desktop personal computers.

Our institute's policy for open-source is similar to the prevailing in the industry. We are using open-source development tools and operating system but we protect the sources of homegrown core processing libraries. At the time there is still a discussion on what business model is appropriate for

making profit from development of open source solutions. In the near future the area that deserves most attention is using open standards for data exchange integration of medical systems. An example is the OpenECG initiative [9]. Still the barrier is the reluctance of large medical companies to conform to open standards for data exchange or to publish specifications of their own. For growth of such standards the pressure from governmental bodies is necessary.

2. HARDWARE AND SOFTWARE

The core of the electrocardiograph is a personal computer with built-in USB interface to connect the external ECG module. The ECG is acquired from the patient's body through electrodes connected the ECG module (containing amplifiers, A/D converters, the defibrillation and pacemaker impulses detection module) and transmitted to the computer using the USB transmission. The Figure 1 presents a diagram of the hardware and the external modules.

The following software components are used:

– Operating system Linux 2.4.19 (with low-latency patches),
– Python 2.2 used in the main application and lots of additional tools,
– QT 2.3 and PyQT 3.6,
– Portable C++ modules (accessible through SIP or BPL) in signal processing, device drivers and ECG widgets.

The Figure 2 presents an exemplary screenshot of the exercise test application.

3. ARCHITECTURE OF DIGITAL SIGNAL PROCESSING SUBSYSTEM

The fast and powerful *Digital Signal Processing Toolkit* have been developed. They allow creating and managing the signal processing networks in Python. In Fig. 3 an overview of the ECG processing in the electrocardiograph has been shown. The Table 1 shows measured times of simple processing networks consisting of 1 or 3 processing blocks (MovingAverage(n) = averaging of n samples, test data stream: 12 channels of 7500 samples, Linux based Pentium 3 1000MHz PC).

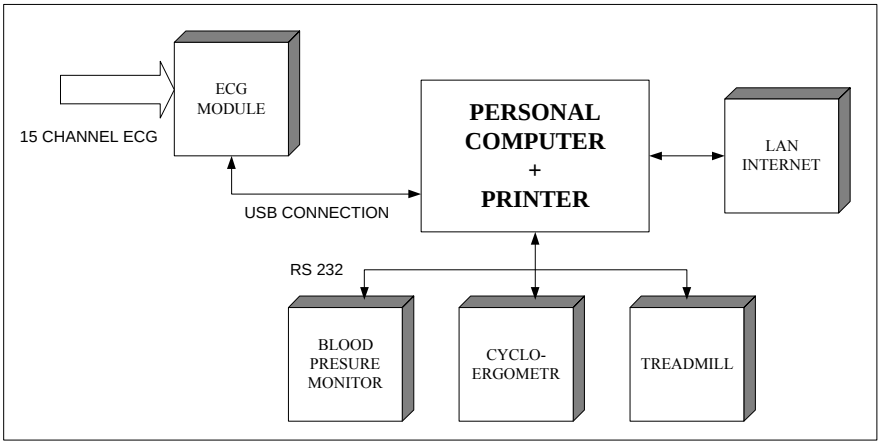

Fig.1 Hardware and external modules diagram.

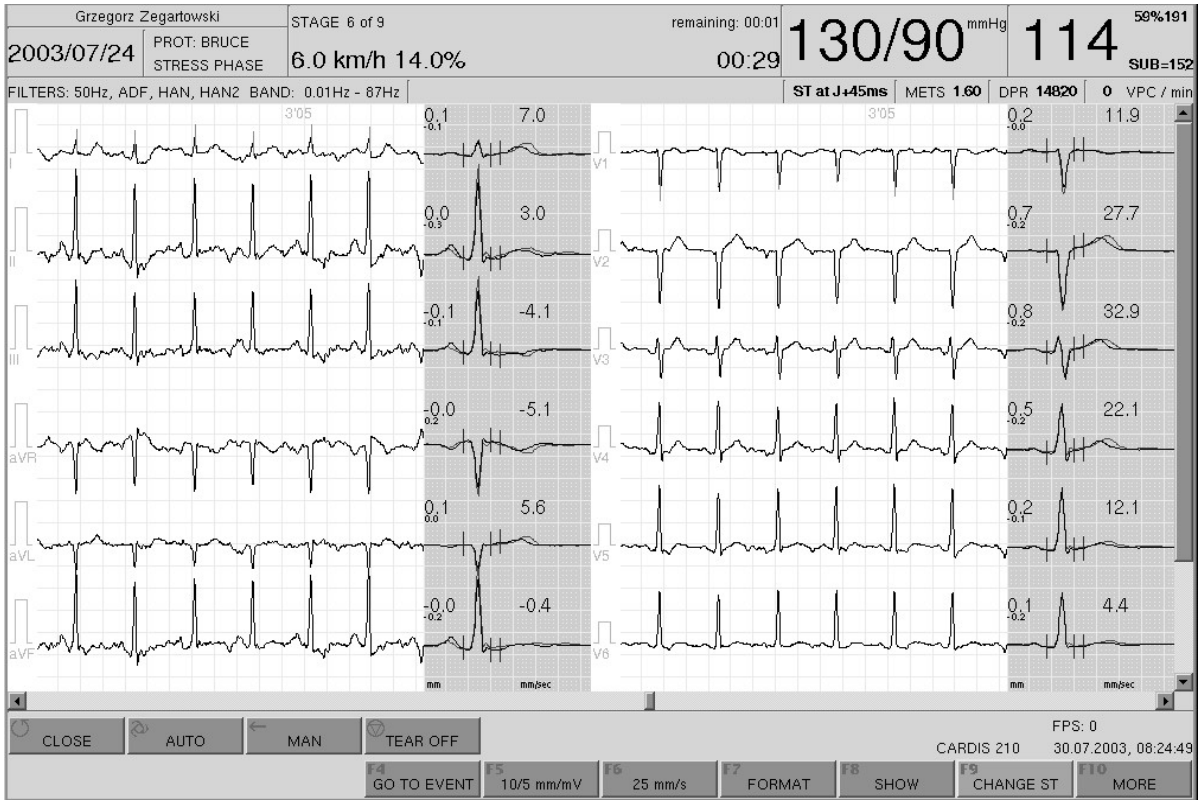

Fig.2 The view of the screen during the exercise test.

Table 1 Measured times of simple processing network.

t1	MovingAverage(10)	1.62 s
t2	MovingAverage(20)	3.04 s
t3	a chain of 3 averages: MovingAverage(10)-> MovingAverage(10)-> MovingAverage(10)	4.84 s

The development tasks of the QRS complex detection and classification algorithms belong to the research tasks of the authors' Institute. The ECG analysis is noticed as a challenge of the ECG signals quality with its low amplitude (range of mV) can be easily interfered by other physiological signals coming from electrical equipment. Furthermore, the ECG acquisition is inhibited by high impedance of the signal's source [1, 3, 4].

The two major types of ECG noise artefacts were distinguished, with:

- frequency spectrum overlapping frequency spectrum of the ECG in narrow range:
  - baseline drift,
  - modulation of ECG with respiration,
  - power line interference,
- frequency spectrum overlapping frequency spectrum of the ECG in wide range:
  - muscles contractions,
  - noise generated by electronic devices,
  - electro-surgical noise.

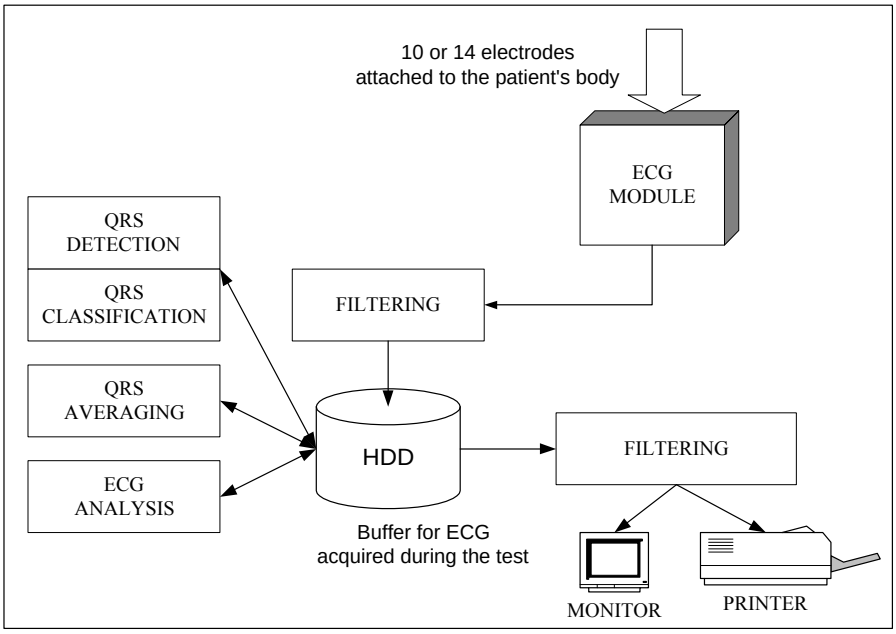

Fig.3 Overview of the ECG processing in the electrocardiograph.

The data interferations with noise signals have to be either filtered or discarded. That is why modern electrocardiographs use various methods of analogue and digital signal processing technologies. However, sufficient bandwidth of an ECG recorder plays the crucial role in accurate acquisition and reproduction of ECGs, which is important in diagnosis making process that comprises automatic measurements and their interpretation. Good high frequency response guarantees appropriate reproduction of the Q-, R-, S- waves and other details within the waves, while good low frequency response provides accurate reproduction of the ST−segment (both in level and slope) [2]. In Fig. 4 the block diagram of the QRS detector and classifier has been shown. That was presented later in details.

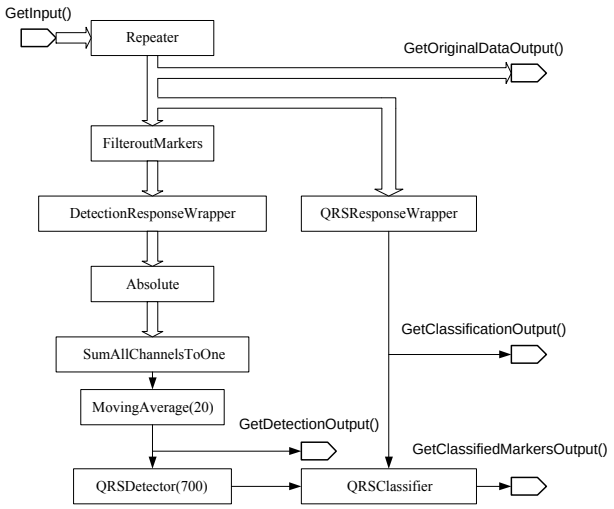

Fig.4 The QRS detection block diagram.

Based on our experiences of ECG processing applications we were able to separate data types and interactions of DSP and to create a toolkit suitable for the production processes.

The following data types were used:

– stream of frames,
– markers of a structure, containing information about characteristic types and time of occurrence, used for storing detected pacemaker impulses, for indication of QRS complexes,
– metadata frames with additional information about the ECG stream.

Based on the data types and on very simple interfaces for input, output, random access buffers, we were able to hide technical details of transmission in configuration scripts (Fig. 5).

```
class QRSDetection:

    def_init_( self ):

        # preparing processing blocks
        self.inputBlock=dsp.Repeater()
        self.filteroutMarkers=dsp.FilteroutMarkers()
        self.detectionResponseFunction=dsp.DetectionResponseWrapper()
        self.absolute=dsp.Absolute()
        self.sumChannels=dsp.SumAllChannelsToOne()
        self.movingAverage=dsp.MowingAverage(20)
        self.classificationResponseFunction=dsp.QRSResponseWrapper()
        self.QRSDetector=dsp.QRSDetector(700)
        self.QRSClassifier=dsp.QRSClassifier

        # build connections
        self.inputBlock.Connect( self.filteroutMarkers )
        self.filteroutMarkers.Connect( self.detectionResponseFunction )
        self.filteroutMarkers.Connect( self.classificationResponseFunction )
        self.detectionResponseFunction.Connect( self.absolute )
        self.sumChannels.Connect( self.movingAverage )
        self.movingAverage.Connect( self.QRSDetector )
        self.QRSDetector.GetmarkersOutput.Connect( self.QRSClassifier )
        self.classificationResponseFunction.Connect( self.QRSClassifier )
```

Fig.5 An example of class in Python.

The research works provided us with signal processing visualisation tools as well. In Fig. 6 the configuration tools, for presentation of simultaneous data streams, were presented - found in the QRS detection and classification network. In Fig. 7 the program results are presented.

The first and the second channel are the original ECG signal obtained from AHA database. The third channel presents the calculated detection function with markers denoting fiducial points. Additionally, the results of classification are presented with appropriate marker (here as for example: VPC – ventricular premature complex or SVPC – supra-ventricular premature complex). We can observe as many channels as it is necessary. We can choose them manually from the right panel.

```
class ShowQRSDetectionNet( Network):

        def_init_( self ):
                Network.__init__( self )

        def createNet( self ):
                self.detector=QRSDetection()
                self.source.Connect( self.detector.GetInput() )

                self.detector.SetParam( "detectionRadius", 8 )

                self.AddOutputToPresentation( self.detector.GetOriginalDataOutput() )
                self.AddOutputToPresentation( self.detector.GetDetectionOutput(),
                self.detector.GetClassifiedMarkersOutput() )
                self.AddOutputToPresentation( self.detector.GetClassificationOutput(),
                self.detector.GetClassifiedMarkersOutput() )
```

Fig.6 An example of using the QRSDetection class.

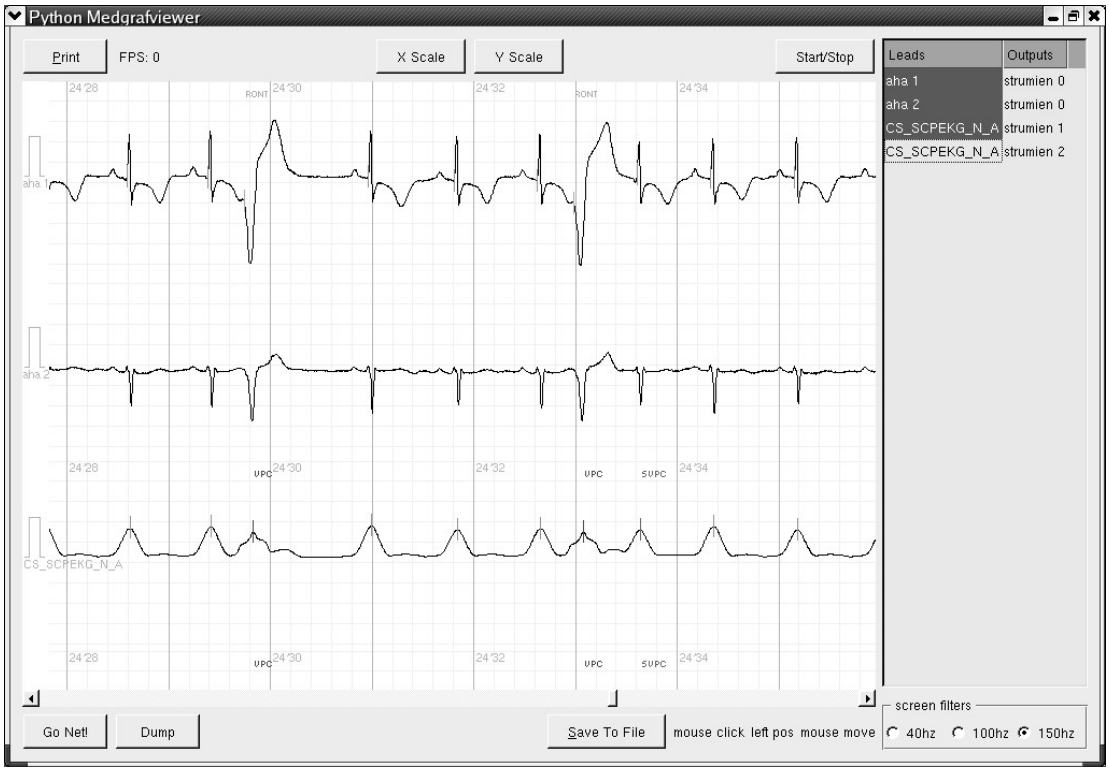

Fig.7 Screen view of the PythonViewer.

4. WHY PYTHON?

We are using Python as the language for development of medical software because it offers:

- Much more efficient development than C++,
- Portability,
- Good binding to QT and PyQT,
- Possibility to write powerful and elegant code,
- Object oriented style,
- Easy re–use and modular design.

Python is a high level script language. Such languages' purpose is often to be glue to other programs or components. Python differs from classical programming languages like for example C++. Whereas in C++ very strong rules of type checking are enforced – during the compile phase – Python checks types of operands to any operation at the moment of its execution. It is also very powerful: one can easily add member functions and data fields during the execution, variables do not have fixed type and memory not used any longer is freed automatically by the garbage collector.

Although execution speed of pure computational code in Python and C++ can hardly be compared – processing overhead of languages like Python is huge in comparison to almost none in C++ – in many cases it is not the most important point.

Today's programs are different from those from the pre-GUI era. Nowadays incredible processing capacity of desktop computers is wasted most of the time and in opposite is not sufficient during short peak load periods – even for optimised C++ applications. For regular GUI applications the performance of programs written in Python is in many cases felt by the users the same way as of programs written in classical, compiled languages.

Still there are the areas, where maximum computational speed is needed. In these areas the typical attitude is to write low level processing procedures in languages like C++ and to integrate these submodules with Python programs. There are already hundreds of common available libraries that might be seamlessly used in Python. This makes Python appropriate for uses in almost all areas of information processing world ranging from astronomy to financial or medical applications [5, 6].

In case of our electrocardiograph we followed the same way. The most demanding parts of the program are implemented in C++. An example is the DSP library. It is designed to allow maximum performance as measured in number of samples processed per second. The functionality of the library is exposed to Python in a seamless way through one of C++ to Python bindings called SIP [10]. The library is used by the main electrocardiograph's application that is written in Python.

What do we get in exchange? The research [7, 8] and the practice show, that the development of programs in Python is much more efficient than in lower level languages like C++. In almost all parts of development process there are advantages in using Python. It is easier to configure the development environment. Through interactive prompt one can instantly check language constructs and functionality of the standard libraries. Expressing programming ideas in Python takes usually much less lines of code than in C++ and vast standard library provides solutions for complex problems and access to many technologies. All that makes Python rather a RAD (Rapid Application Development) tool than a programming language.

All core software components that are the basis for electrocardiograph's specialised programme are open source (Linux, Python, Qt, PyQt, gcc, to name a few). The main advantage of using open source tools is the ability to fix bugs in those tools in-house. It is also possible to extend

their functionality by providing features filling specific needs. By studying the sources one can also get better understanding of internal mechanisms and thanks to that develop more robust programmes. Open source tools are often portable; thus they do not tie developers to a specific hardware platform or software and system solutions.

There is also one other aspect of open source: it stimulates using open standards for data exchange. It does not make sense to keep secret a file format or the details of a communication protocol implemented by open source solutions as it often happens in the world of closed source.

5. CONCLUSIONS

The system has proven its ease-of-use and performance. It has become relatively easy to combine different ECG processing methods using Python driven and C++ based DSP library, supporting signal processing engineers with neat graphical user interface. Also through tests have confirmed that new flexible approach based on combination of Python and C++ adds minimal and acceptable computing overhead when compared to previously used hard-coded C++ template based libraries. Altogether the solution has shown flexibility and efficiency and validated further use of free open-source technologies like Python and Linux operating system.

BIBLIOGRAPHY

[1] FRANKIEWICZ Z., ŁĘSKI J., PAWŁOWSKI A., Wybrane zagadnienia cyfrowego przetwarzania sygnałów biomedycznych. Skrypt Uczelniany, Nr 1705, Wydawnictwo Politechniki Śląskiej, pp. 9-15, Gliwice, 1993.
[2] CDV – Committee Draft for Voting, Draft IEC 62D: 60601-2-51/Ed.1. Medical Electrical Equipment, Part 2 – 51: Particular requirements for the safety, including essential performance of recording and analysing single channel and multi-channel electrocardiographs. 15 March 2001.
[3] American National Standard Institute, Diagnostic electrocardiographic devices. ANSI/AAMI EC11-1991.
[4] GIBIŃSKI P., OWCZAREK A., Assessment of low-frequency response of ECG recorders in relation to international requirements. Journal of Medical Informatics & Technologies, Vol. 4, pp. 63-66, University of Silesia, November 2002.
[5] ALTED F., Processing and analysing very large amounts of data in Python. EuroPython Conference, Charleroi, Belgium, 25 June 2003.
[6] LUTZ M., ASCHER D., Learning Python. O'Reilly, pp. 9-25, UK, April 1999.
[7] PRECHELT L., An empirical comparison of C, C++, Java, Pearl, Python, Rexx, and Tcl for a search/string-processing program. Technical Report 2000-5, 34 pages, Universität Karlsruhe, Fakultät für Informatik, Germany, March 2000.
[8] SHAFER G. D., Python is the enterprise. Builder.com, *http://insight.zdnet.co.uk/software*, July 2002.
[9] Christoph Zywietz (BIOSIGNA) and Ronald Fischer (University of Hannover) How To Implement SCP-ECG, Part II. *http://www.openecg.net/SCP_howto_II.pl*
[10] RIVERBANK COMPUTING LTD, SIP – Overview, *http://www.riverbankcomputing.co.uk/about/index.php*