Development and Implementation of Software Architecture for Data Visualization With Various Medical Devices

I.N. Rodionov[1], I.V. Nesterenko, D.V Telyshev
Department of biomedical system
National Research University of Electronic Technology
Zelenograd Moscow, Russian Federations
[1]rodionov@bms.zone

T.G. Le
Bacoulev Institute for Cardiovascular Surgery
Moscow, Russia
tanya_co@mail.ru

***Abstract*—The software package capable to provide support of various medical devices has been developed. Architecture of the program is based on the "Mode-View-Controller" pattern, with a passive model. Each module implements the structural pattern "Bridge". At the expense of it the system of plug-ins is provided. Plug-ins for the "Model" form the data from various devices. Plugin "View_Qt" implements a graphical user interface and provides centralized access to data. Data filtering happens at the level of model, but their check is carried out in the controller.**

Keywords— biomedical; pattern; rogramming; design; software

I. INTRODUCTION

Rapid development of the modern technique allowed to expand considerably a line of the medical equipment applied in medical institutions. However, such a variety of devices has led to the emergence of a large number of complex software systems, which need to work doctor. An example may be a program for analyzing ECG and EEG data, the program of information processing about operation of artificial heart. They all have similar functionality, but they can only work with one specific device. For lowering of time necessary for training on the medical equipment is offered the universal software allowing not only to reduce time for training of medical staff, but also to reduce costs of software development for the specific equipment (Fig. 1).

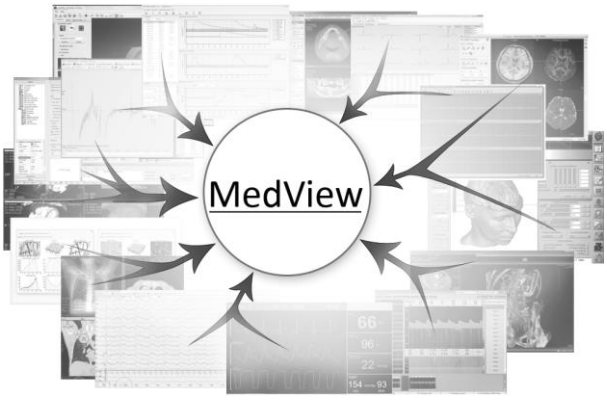

Fig. 1. Illustration ideas single data access medical devices

II. PAGE STYLE

The basis of the program architecture is a pattern Model-View-Controller (MVC) [1]. Its main feature is that the model of application and the user interface are partitioned into separate components so that a change in one of them has a minimal impact on the other. There are several varieties of MVC. Among them the scheme with passive model was selected, i.e. the model has no opportunity to influence representation (Fig. 2).

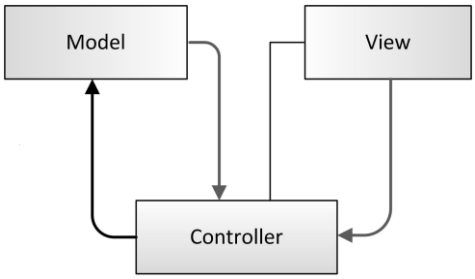

Fig. 2. Scheme pattern "Model-View-Controller"

In order to ensure greater flexibility of architecture it is decided that each of MVC modules realizes a pattern "Bridge" (Fig. 3) [2]. Its essence lies in the fact that the basic module is an abstraction, whose main task is redirection of all the received requests to other elements, in this case to dynamically connected elements – plug-ins. In turn in plug-ins are implemented special cases for each of the possible queries.

We will consider it on the example of the Model module (Fig. 4). In it there is a method "fillControllerDate()" – responsible for filling of data for the controller, in which is called the appropriate method of the internal pointer and the returns result of its operation. This internal pointer is a virtual function table [3], which is the interface, with which operates the Model, and an interface that should override all plug-ins for the model.

Currently developed a plugin for "Model_Heart" model, which is responsible for access to the data generated during the operation of the device of the long mechanical substitution of function of heart. It defines the necessary interface for the model, including method "fillControllerDate()" in which data for the controller are formed from the data read from the file.

978-1-5090-0445-4/16/$31.00 ©2016 IEEE

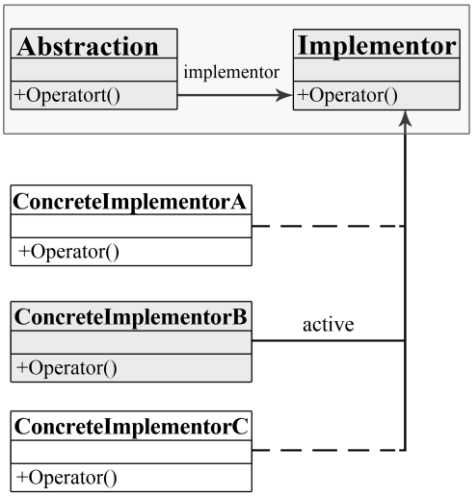

Fig. 3. Scheme structural pattern "Bridge"

In summary, using different plug-ins, depending on model parameters, the user obtains data from different devices. Moreover, data can be taken both from the file, and from a remote server, or to be read out directly from the device. As a result to add support of the new device it is necessary to write the appropriate plug-in for model. At the same time it isn't necessary to recompile all system since plug-ins are connected dynamically – i.e. in the course of operation of application.

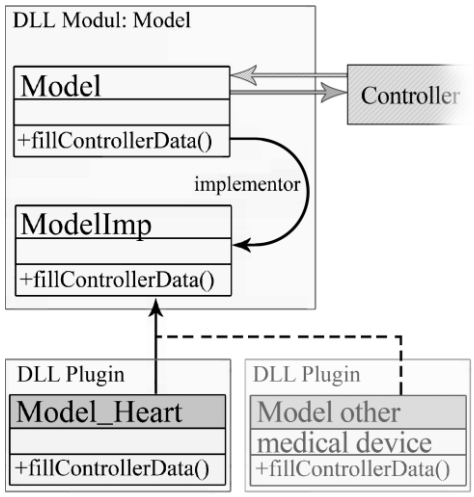

Fig. 4. Implementation of the "Bridge" pattern on the example of "model" module

Each of modules realizes the "Bridge" template, i.e. they supports system of plug-ins. Including the module "View" for which plug-ins are responsible for the user interface. One of them is "View_QT" – the plug-in based on a framework of Qt [4]. The area of data mapping which is available in it is based on the "QtCharts" library, designed to draw graphs. The majority of similar libraries work by the following principle: in the beginning to them all available data array is provided, further there is their processing and display, and only then the user can interact with available information [5]. The problem of this scheme is that there is a duplication of information. These models may have changed over time, but because stored in the interface of the old data copy, then it does not reflect the changes occurred with the data. It was therefore decided to use other algorithm of operation. Every time there is a change in the data display area: the application window size is changed, changed the scale, triggered a timer update - this field is redraw. But before this new data is requested from the model for the new desired area. Only on the basis of the newly received data is repainted. Model offers not all data, but only a part. Selects the available information, based on the parameters of the request. It provides data only from the required area - ie what a user wants to see at particular time.

But happens that there is too much data for necessary area in model. For example if conditionally to take area of 100x100 pixels on a display element of diagrams, then for this period the model can have 10 000 samples (Fig. 3). But for obvious reasons we could display only 100 points. Therefore in model there is data filtering. on the basis of different methods – decimations or approximations. Data is actually formed for the specific screen resolutions. For this example on an output receive 100 points which need to be displayed on a graphics. Besides, always realized is one case - the complete data update.

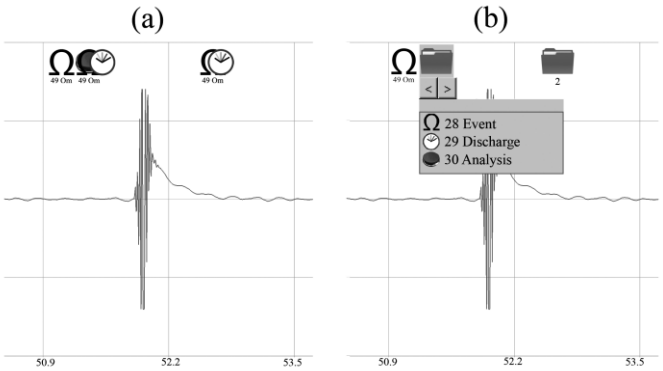

Fig. 5. (a) Superimposing event icons on each other; (b) Packing events close group and access group data

In the provided scheme, the model completely is responsible for information obtained by the user. Therefore at the different scale it is possible to show the different or changed data. For example, "events" - their icons are displayed in the upper part of the area of visualization of data. At the certain scale of an icon will strongly superimpose each other (Fig. 5.a), worsening their perception. Therefore the model assembles events into groups and presents them in the form of one event, when you click on that you can is possible to receive the complete list of the hidden data (Fig. 5.b). Uploading of contents of group occurs only while the user demands this information.

As a result, this system makes it easy to work with large amounts of data, which is essential, as the target medical equipment, for which developed the software package: sensors ECG and EEG, the device long mechanical replacement heart function - are high-frequency devices that generate data with a large frequency and, consequently, in large volumes.

In the controller, and specifically in its plug-in of "Controller_Standard" there is a data conversion created by "Model" into a form perceivable by "View" and also

transmission of requests from "View" in "Model" (Fig. 6). In addition in this module there is additional data filtering as well as correction parameters incoming requests [6]

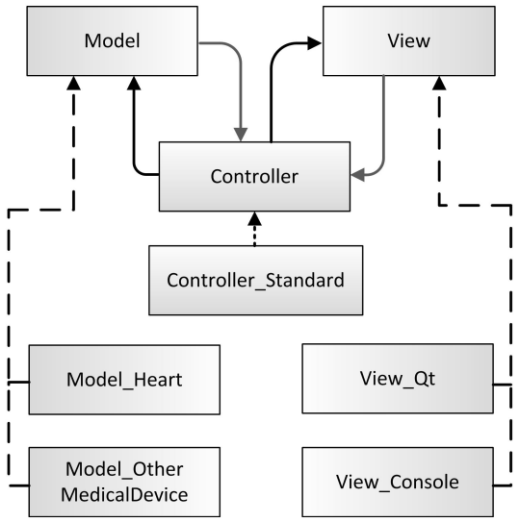

Fig. 6. The final scheme of architecture of the developed program complex

III. Conclusions

As a result, we get quite a flexible architecture in which each element of the MVC is implemented through a plugin system (Fig. 6). This makes it easy to add support for new devices, and secondly allows multiple user interfaces. For example, in addition to "View_Qt" is created the plug-in for the console interface – "View_Console". So far it is used for testing of the developed system, but in the future it can work with third-party software, which only the data from the instruments will be required, without the graphical user interface.

The present software allows you to view data received from various devices (Fig. 7). This will significantly reduce the time of training of medical personnel, as will no longer need to work with a number of programs designed for specific devices.

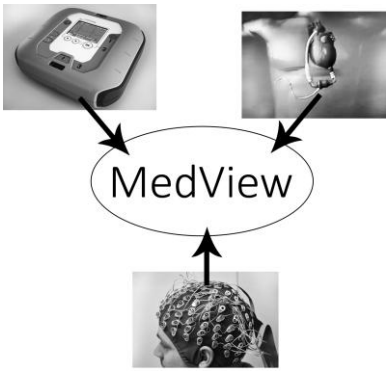

Fig. 7. Devices which are planned to be united with the help of the developed program complex

Markedly reduced the time to create software for the new hardware. Developers will need to write only a part of the system - a module that provides access to specific data, and then connect it to the submitted software.

Acknowledgment

The work was supported by the Ministry of Educational and Science (project ID RFMEFI60715X0113).

References

[1] M. Fowler, *Patterns of Enterprise Application Architecture*, 1st ed.: Addison-Wesley Professional, 2003.

[2] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns*: Addison-Wesley, 1994.

[3] B. Stroustrup, *The C++ Programming Language*, 4rd. ed.: Addison-Wesley, 2013.

[4] M. Schlee, *Qt 5.3 Professional programming in C++*, St. Petersburg, Russian Federation: BHV-Petersburg, 2015.

[5] J. Kerievsky, *Refactoring to Patterns*: Addison-Wesley, 2004.

[6] E. Freeman, *Head First Design Patterns*: O'Reilly Media, 2004.