International Journal of Applied Engineering Research ISSN 0973-4562 Volume 10, Number 17 (2015) pp 38557-38561

Basics of QtQuick technology and its application in the development of mobile multi-function hardware-software of long-term cardiomonitoring and ergonometry

Ivan Alexandrovich Grinko
Scientific and Technical Center "Technocenter" Southern Federal University,
347900, 81 Petrovskaya street

Abstract- The article discusses the use of the latest technology QtQuick, allowing to create cross-platform applications using declarative language QML, and it`s application during development of multi-functional mobile hardware-software complex and long-term cardiomonitoring and ergonometry. QtQuick was first introduced in Qt 4.7 and allowed to develop programs for the operating systems Windows, Linux, Mac OS, Android, iOS and WindowsRT. This technology has made possible the creation of multifunctional dynamically changing graphical user interface with visual effects and animation. Focused on the use of mobile devices, touch input. Includes QML language, based on JavaScript, it`s interpreter and a collection of ready components. QML program is a set of files, containing QML elements, organized in trees, representing a set of graphic and behavioral units, such as states, transitions and animation. Items can be combined into complex components with spread input output interface. QML objects functionality can be extended with JavaScript, and added to project as standard .js files. To provide even greater flexibility and transparency when creating applications, you can use C ++ through the Qt framework. QtQuick has mechanisms for two-way integration with C ++ objects, including supporting of Qt signals and slots mechanism. Thanks to its capabilities, QtQuick technology was chosen to implement the application for debugging hardware modules of multifunctional mobile hardware-software complex of long-term cardiomonitoring and ergonometry. The program was ported from Windows to Android without any effort.

Keywords: Qt, QtQuck, QML, JavaScript, C ++, GUI.

Introduction

Qt is a cross-platform framework for the creation of applications and the user interface [1]. During the existence from 1996 year Qt has changed several owners, has got a huge community of developers and has noted using in software products of such firms, as Adobe, AT&T, Cannon, HP, Bosch, IBM, Motorola, NASA, NEC, Pioneer, Sharp, Siemens, Sony, Xerox and others [2].

The application written by the use of Qt can be started in the majority of modern operating systems by simple compilation of a program without changing a primary code [2]. From other libraries Qt differs by existence of the special metaobject compiler (MOC) allowing to expand opportunities of C ++, to provide a convenient system of the interobject connection through the mechanism of signals and slots to developers and to organize an association of objects in hierarchy for obtaining the expanded information about their nature during the implementation of a program and the automatic control of memory [2].

Qt is divided into modules consisting of a set of classes which functionality covers a great part of opportunities of modern operating systems significantly making the development oa applications easier. Qt is easy expanded and supports a technology of component programming [2].

Originally Qt was aimed at providing developers of the desktop software with simple and powerful tools for fast creation of cross-platform applications [3]. However, a tendency of replacement of classical personal computers by portable laptops and mobile devices [4] is now observed. In this regard methodologies of the development of the user interface also change.

The main parte

To satisfy the created requirement a new QtQuick technology allowing creating difficult components of the interface with support of sensor input [5] was presented in the Qt 4.7 version. It allowed applications on Qt to change desktop operating systems to mobile ones having provided a convenient way of the work with programs for both worlds.

Qt 5 became a full revision of the architecture, which was earlier and concentrated on the following opportunities [6]:

- The use of graphics based on OpenGL (ES) with support of a scene graph for creation of visual effects;
- The injection of the QML and JavaScript use for the user interface. Thus, the development of the application can be divided into the front-end (QML + by JavaScript) and the back-end part (C++) which on one hand will allow to create modern and attractive programs without the loss of productivity;
- The expansion of support of mobile systems includes not only a possibility of Qt on Android, iOS and Windows applications launch, but also the simplified mechanism of porting of programs on other operating systems.

QtQuick is based on several technologies, among which are [6]:

- QML - the declarative markup language for creation of the user interface;

- JavaScript – the scripting language for description of the operation logic of the interface;

- Qt C ++ is a set of libraries on C ++ for the back-end part of the application.

Parts of the application on QtQuick are transmitted and placed in memory as standard for Qt C ++ objects. Thus, the loss in productivity is observed only on the start of a program and can be leveled by preliminary compilation of QML objects [7].

Like HTML, QML is also the markup language and consists of a set of special tags. Such decision allowed applying a declarative approach to creation of the user interface, to make it easier and more evident [8]. The functional opportunities of QML objects extend at the expense of JavaScript. C ++ [6] is used for realization of parts of the application which are beyond QML/JavaScript.

QML elements completely support the mechanism of signals and slots of Qt for the interobject connection and supplement its binding of properties. The QtQuick platform automatically traces the connected sizes and causes the following changes with properties of objects [9].

All QML elements can be united in groups according to carried out functions by them. Into a basic group of elements enter [6]:

- Item – is a basic element for all visual components. It does not contain drawing functions but provides properties, which each visual component [10] must possess.

- Rectangle - is the Item expansion that allows to fill the area by a continuous color or a gradient and to outline its borders. It is possible to draw not only rectangles, but also circles, manipulating by "height", "width" and "radius" [11] properties.

- Text – is a submission of the text. The sizes of an element are calculated from the text containing in it. All aspects of a display are changed.

- Image - the image. All available formats of pictures in Qt are supported.

- MouseArea - realizes support of the user input using a mouse or gestures on sensor devices.

All visual QML components are constructed from the described ones above. In addition, they can process various user commands.

With these objects, it is possible to make various manipulations including scaling, rotation, and movement. Elements can be grouped in lines (Row) and columns (Column). Relatively to each other, the objects can be positioned with the use of "anchors" properties [6].

The following group includes more complex elements for user input processing [6]:

- TextInput – is used for the text input. It is possible to operate the input using standard validators or own based on regular expressions.

- FocusScope – controls over the input focus transfer.

- TextEdit – is a multirowed text input with additional opportunities of the text decoration.

- Keys – is processing of events of the input from the keyboard.

Further, the elements, visualizing a change of the property from a primary value to the final one, which is animation, will be considered. It is possible to define nature of a change of sizes by special functions. The main group of available animations contains [12]:

- PropertyAnimation – the animation of a change of properties;

- NumberAnimation – the animation of a change of sizes;

- ColorAnimation – the animation of color;

- RotationAnimation – the rotation animation.

The complete list of animation objects is much wider and this article will not consider it.

EasingCurves - will define the mathematical law according to which the animation will be applied. There are 10 basic laws; the linear one is used by default. Animations can be united into consecutive and parallel groups, stopped and started at any moment of playing [6].

It is often convenient to describe the user interface in the form of a set of states - a set of properties, which can be changed by any event. Transitions between properties can be animated. It is also possible that states themselves react to events with the use of signals and slots [6].

In QML the state is defined with the help of the "State" element containing a corpus of properties and values, conditions of transition to other states and handlers of certain events. A special machine unites the states and controls over their change [6].

QML supports Model-View-Delegate template - assuming that data are stored separately from their representations. The model defines what data will be displayed and how they can be changed [6]. During application programming it is very important to understand a need of the separate data storage and the way of their representation. It does the system flexible and expanded.

The model (Model) in Qt ideologies presents a set of data; representation (View) is their visual display. The delegate (Delegate) provides a connection of the model and representation, operating the model visualization. The basic representations in QML are [6]:

- ListView – is the representation in the form of a list;

- GridView – is the representation in the form of a grid.

Representations may contain headings, various displays of active and inactive elements, can process the pressing of keys and gestures for the organization of navigation [13].

If the representation points a general concept of visualization of data, than the delegate points a direct display of data. The delegate, having a direct access to the model of data and representation, can operate a visual display and editing data [14].

Representations can animate an addition and a removal of data from the model with the use of definition slots onAdd and onRemove [6].

QML can use the standard models of Qt data or created by the user.

Though initially QtQuick does not contain elements of management, the type of a button, combobox, checkbox, they can be realized independently without any difficulties. In addition, the company Digia that now owns [15] Qt, created a

set of QtQuick Controls widgets for filling this gap. The use of these elements guarantees the interface components under control of various operating systems [1].

If there are not enough opportunities of QML for realization of necessary functions, Qt C ++ steps on the stage. The first task in this field is a connection of QML and C ++ objects. The further matter will be about it [16]. It should be noted now that the ways given below are not only right.

Because all QML types are successors of a QObject class, the QtQuick platform can use the metaobject Qt system for the bilateral interaction between objects.

QML components form hierarchical trees, an access to concrete object can be got using QObject:: objectName property and QObject:: find() function. Then for reading and writing down the properties the QObject functions QObject::setProperty() и QObject::property() can be used [17].

Any QML method can be executed from C++ using the QMetaObject::invokeMethod() function. The signature of this function demands that input parameters and output values are the QVariant type because exactly it is used for transfer of data between QML and C++ [17].

All QML signals are automatically available from C++ through the mechanism of a standard QObject:: connect() connection. Similar to any C++ the signal can be accepted by QML object. As usual, signals can transmit data as its arguments [17].

QML can directly work with the C ++object. The following requirements must be done for this purpose [18]:

- C++ class must be the successor of QObject and contain Q_OBJECT macro;

- Available for QML methods must take place in the section “public slots” or have the Q_INVOKABLE modifier;

- The properties must be made out as Q_PROPERTY;

- The entrance and output data must be the following [19] (table 1);

Table 1. The entrance and output data

Qt type	Coordinated QML type
bool	bool
unsignedint, int	int
double	double
float, qreal	real
QString	string
QUrl	url
QColor	color
QFont	font
QDate	date
QPoint, QPointF	point
QSize, QSizeF	size
QRect, QRectF	rect
QMatrix4x4	matrix4x4
QQuaternion	quaternion
QVector2D, QVector3D, QVector4D	vector2d, vector3d, vector4d
Enumerated, formalized as Q_ENUMS()	enumeration

QVariantList and QVariantMap will automatically be transformed by QML cursor into the JavaScript array or the object. The same also refers to QDateTime. Without problems the following Qt containers will be transformed: List <int>, QList <qreal>, QList <bool>, QList <QString>, QStringList, QList <QUrl> [19].

- C++ class must be registered in the QML system with the use of the qmlRegisterType() function [20].

A set of the taken measures allows C ++ objects to look and behave themselves as usual QML elements.

Even from such familiarity with QtQuick and QML it is possible to make a conclusion that this technology contains a big set of techniques and opportunities for creation of a wide range of the cross-platform software. Exactly for this reason, it was chosen for implementation of the application for debugging of modules of a mobile multipurpose hardware-software complex of long cardiomonitoring and ergonometry.

The development of the application for testing of hardware modules taking into account a possibility of installation and start of a program not only under Windows, but also under Android entered a task. A connection with modules is carried out on Bluetooth.

It was decided to give a chance for QtQuick to prove itself, the creation of Cardiobugger application, which is used by developers for debugging of hardware modules, was the result, and it is possible to do it with Windows or Android devices. The results shows that QtQuick and QML are really powerful tools of the creation of modern applications. Figure 1 gives the screenshot of the program work. However, there were some difficulties about which the further matter will be.

QtQuick in Community version does not contain widgets for drawing graphs [21]. The problem is solved by the creation of own element based on QMLCanvas - a component for drawing simple and difficult figures and images displaying. An addition, text, shadows, gradients and low-level operations with pixels are also available. Programming is conducted on JavaScript on available QML Canvas functions corresponds HTTP5 Canvas [22]. The code of drawing needs to be placed in the onPaint method for copying of a context to cause the Canvas::requestPaint() function.

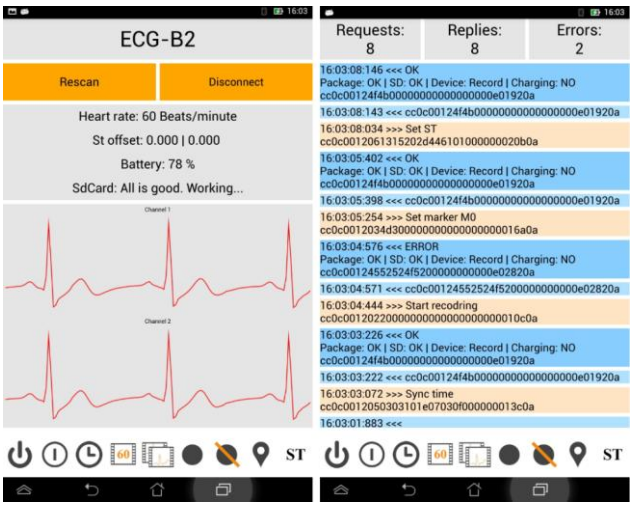

Fig. 1. The work of Cardiobugger on Android

The following problem was the lack of Bluetooth support at Qt on Windows (in 5.4 version) [23]. To resolve the issue for a start the abstract IBluetoothProvider class for granting the unified interface of the interaction with other objects was created. A class for the work with Bluetooth on Android - AndroidSppProvider was inherited from IBluetoothProvider and uses internal Qt Bluetooth methods. A class on Windows - WindowsSppProvider which is also inherited from IBluetoothProvider used WindowsAPI for the work with Bluetooth. During the registration of a class in QML system, the type of the operating system was defined, and the necessary one was changed. Figure 2 gives the example.

```
#ifdef Q_OS_ANDROID
qmlRegisterType<AndroidSppProvider>("com.ivware16.components"
, 1, 0, "BluetoothProvider");
#else
qmlRegisterType<WindowsSppProvider>("com.ivware16.
components", 1, 0, "BluetoothProvider");
#endif
```

Fig. 2. Registration of a class for the work with Bluetooth

Thus, depending on the compiler under the BluetoothProvider name either WindowsSppProvider, or AndroidSppProvider was registered. No additional manipulations were required.

By similar way the switching of elements representations depending on the operating system was realized (fig. 3).

```
    #ifdef Q_OS_ANDROID
    qmlRegisterType(QUrl("qrc:/MobileLayout.qml"),
"com.ivware16.components", 1, 0, "ElementsLayout");
    #else
    qmlRegisterType(Qurl("qrc:/DesctopLayout.qml"),
"com.ivware16.components", 1, 0, "ElementsLayout");
    #endif
```

Fig. 3. Registration of the apportion of elements

It was a necessary step because the desktop layout contained widgets which representation is inconvenient and inexpedient on mobile devices. Therefore, it was decided to make two configurations with the unified interface of input-output, but different elements. A necessary class was chosen at a compilation stage depending on the used compiler.

The following problem consisted in particular works of Bluetooth on Android devices. On Android 4.2 version, it was revealed that if the application is kept in a cache, but the connection with Bluetooth module was not broken off, there was a risk of impossibility of the further work with Bluetooth without a reset of Android device [24]. Thus, it was necessary to control the condition of a program, and at its preservation in a cache and the further closing, to break Bluetooth connection off. It cannot be done with the use of QML, but it is possible in C++.

QGuiApplication contains the signal informing about the change of the application status - QguiApplication:: applicationStateChanged(), it is only necessary to transfer these data to root QML object. Not to encumber a code with the excess designs it was decided to use the new syntax of Qt signals and slots working without additional metainformation (fig. 4) [25].

```
    Qlist<Qobject *>qmlObjects = engine.rootObjects();
    Qobject *mainObject = qmlObjects.first();
    QvariantreturnedValue;
    Qobject::connect(&app,
    &QguiApplication::applicationStateChanged,
            [&](Qt::ApplicationState state){
    QmetaObject::invokeMethod(mainObject,
"applicationStateChandges",
            Q_RETURN_ARG(Qvariant,
returnedValue),
    Q_ARG(Qvariant, state));});
```

Fig. 4. Sending the signal of the status change of the application to QML

Thus, the root QML element began to receive notices about the status of the application and closed independently Bluetooth connection if it will be necessary.

Conclusion

Despite so short look at QtQuick and QML, it is possible to safely recommend this technology for the development of big software products with the difficult user interface. Of course, there are problems which are specific for each platform, but one is clearly absolute - Qt applications can be now already compiled under the majority of modern operating systems and provide to their users the equivalent and convenient tools for solution of their tasks.

The results of researches, which this paper has presented, where got with the financial support of the Ministry of Education and Science of the Russian Federation within realization of the "Creation of Advanced technology production on fabrication of a mobile multipurpose hardware-software complex of long cardiomonitoring and ergonometry" project according to the governmental resolution № 218 (9 April, 2010). Researches were conducted in FSAEI of HE of the SFU.

References

[1] Digia. Qt Project [Электронный ресурс] // Qt Project: [сайт]. [2015]. URL: http://qt-project.org/ (accessed date: 29.03.2015).

[2] Shlee M. Qt 4.5 Professionalnoe programmirivanie na C++. Saint-Petersburg: BKHV-Petersburg, 2010.

[3] Blanchette, J. & Summerfield M. 2006, C++ GUI Programming with Qt 4. Prentice Hall. pp. 560.

[4] Freeman, J. 2013, Who nailed the principles of great UI design? Microsoft, that's who // InfoWorld. 2013. URL: http://www.infoworld.com/article/2614315/application-development/who-nailed-the-principles-of-great-

ui-design--microsoft--that-s-who.html?page=3 (accessed date: 29.03.2015).

[5] Qt Project. Vvedenie v Qt Quick // Qt Project. 2011. URL: https://qt-project.org/wiki/IntroductionQtQuick_Russian (accessed date: 29.03.2015).

[6] Johan, T.J. Juergen Bocklage-Ryannel & Johan Thelin. 2014.

[7] The Qt Company Ltd. Qt Quick Compiler // Qt IO. 2015. URL: http://doc.qt.io/QtQuickCompiler/ (accessed date: 29.03.2015).

[8] Zharko, Miyailovich D., 2013, Tehnologii razrabotki polzovatelskih interfeisov // Otkrytye sistemy. URL: http://www.osp.ru/os/2013/10/13039072/ (accessed date: 29.03.2015).

[9] The Qt Company Ltd. Property Binding // Qt IO. 2015. URL: http://doc.qt.io/qt-5/qtqml-syntax-propertybinding.html (accessed date: 29.03.2015).

[10] The Qt Company Ltd. Item QML Type // Qt Documentation. 2015. URL: http://doc.qt.io/qt-5/qml-qtquick-item.html (accessed date: 29.03.2015).

[11] The Qt Company Ltd. Rectangle QML Type // Qt Documentation. 2015. URL: http://doc.qt.io/qt-5/qml-qtquick-rectangle.html (accessed date: 29.03.2015).

[12] The Qt Company Ltd. Usecase - Animations In QML // Qt Documentation. 2015. URL: http://doc.qt.io/qt-5/qtquick-usecase-animations.html (accessed date: 29.03.2015).

[13] The Qt Company Ltd. ListView QML Type // Qt Documentation. 2015. URL: http://doc.qt.io/qt-5/qml-qtquick-listview.html (accessed date: 29.03.2015).

[14] The Qt Company Ltd. Model/View Programming // Qt Documentation. 2015. URL: http://doc.qt.io/qt-5/model-view-programming.html (accessed date: 29.03.2015).

[15] The Qt Company Ltd. Qt Quick Controls // Qt Documentation. 2015. URL: http://doc.qt.io/qt-5/qtquickcontrols-index.html (accessed date: 29.03.2015).

[16] The Qt Company Ltd. Qt QML // Qt Official Site. 2015. URL: http://doc.qt.io/qt-5/qtqml-index.html (accessed date: 29.03.2015).

[17] The Qt Company Ltd. Interacting with QML Objects from C++ // Qt Official Site. 2015. URL: http://doc.qt.io/qt-5/qtqml-cppintegration-interactqmlfromcpp.html (accessed date: 29.03.2015).

[18] The Qt Company Ltd. Integrating QML and C++ // Qt Official Site. 2015. URL: http://doc.qt.io/qt-5/qtqml-cppintegration-topic.html (accessed date: 29.03.2015).

[19] The Qt Company Ltd. Data Type Conversion Between QML and C++ // Qt Official Site. 2015. URL: http://doc.qt.io/qt-5/qtqml-cppintegration-data.html (accessed date: 29.03.2015).

[20] The Qt Company Ltd. Defining QML Types from C++ // Qt Official Site. 2015. URL: http://doc.qt.io/qt-5/qtqml-cppintegration-definetypes.html (accessed date: 29.03.2015).

[21] Digia. Download Qt // Official Qt Site. 2015. URL: http://www.qt.io/download/ (accessed date: 29.03.2015).

[22] The Qt Company Ltd. Canvas QML Type // Qt Documentation. 2015. URL: http://doc.qt.io/qt-5/qml-qtquick-canvas.html (accessed date: 29.03.2015).

[23] The Qt Company Ltd. Qt Bluetooth // Qt Official Site. 2015. URL: http://doc.qt.io/qt-5/qtbluetooth-index.html (accessed date: 29.03.2015).

[24] Young D.G. A Solution for Android Bluetooth Crashes // Radius Networks. 2014. URL: http://developer.radiusnetworks.com/2014/04/02/a-solution-for-android-bluetooth-crashes.html (accessed date: 29.03.2015).

[25] The Qt Company Ltd. Signals & Slots // Qt Documentation. 2015. URL: http://doc.qt.io/qt-5/signalsandslots.html (accessed date: 29.03.2015).