sequality
SOFTWARE
ENGINEERING

Qt QML v HTML5 – a practical comparison

The Sequality bottling demo application

This white paper describes the development work needed to create Qt and HTML5 versions of the same application, and compares the technologies from a technical and economic perspective.

Executive summary

Background

Sequality is an Austrian software engineering company that creates industrial applications, user interfaces for touch displays, and software for embedded hardware systems.

Over the past year, more and more customers had been asking if they should use HTML5 or Qt using the QML declarative UI language to develop software for embedded devices, and wanted to better understand the differences in using the two technologies.

In order to give the most objective advice to their customers, Sequality decided set up a test: give the same developer 160 hours to create a demo of an embedded system using Qt and 160 hours to create the demo using HTML5. These demos would show exactly how the two technologies compare – in terms of development, performance, and sustainability – when used to create the same product. This white paper describes the development work needed to create the Qt and HTML5 demos, as well as the end results.

The developer tasked with creating the demos was experienced with using HMTL5 and C++, but had little experience creating user interfaces using Qt and QML.

The demos were created independently without any vendor input.

Results

The demos showed that although the same amount of development time was spent on both versions, implementation with Qt QMLdelivered a more functional and complete user interface than the HTML5 version. The testing and debugging process was found to be more straightforward with Qt QML, not least because it didn't need testing on multiple browsers.

In general, the Qt QML version responded more quickly and enabled features, like keyboard and multi-touch, that were not supported by HTML5 without additional implementation.

> Over the past year, more and more customers had been asking if they should use HTML5 or Qt using the QML declarative UI language to develop software for embedded devices, and wanted to better understand the differences in using the two technologies"

From an end-user perspective, the Qt QML version behaved exactly as expected regardless of the browser or screen being used to view it. This is because Qt-based applications are compiled for the target, meaning that in terms of user observation, they behave exactly the same no matter which platform they run on. HTML5-based applications, on the other hand, run on the browser of the target, for example Chrome, meaning different platforms can show different behavior as the browser might use different rendering engines depending on the platform.

In terms of the sustainability of the technology, Qt QML is a mature technology (compared to most JavaScript frameworks) that has been developed to ensure backwards compatibility. The AngularJS framework for HTML5 is relatively new, and a valid concern is whether it will be replaced by a new framework in the future. In contrast, QML is very likely to still be supported in 5 years.

Overall, Sequality found that the development of the applications was very different and one needs to carefully consider the benefits and drawbacks of each technology before deciding which one to use.

If the outcome of such an evaluation does not show major advantages of a particular technology, we would recommend Qt over HTML5. In our showcase, the Qt-based application was generally faster, more responsive, and easier to implement.

The bottling demo

Application requirements

The bottling demo application represents a fictional bottling plant supervisory control and data acquisition (SCADA) system. The dashboard-like user interface contains most of the user interface (UI) elements that are common in modern applications.

The application was designed to visualize the process at the bottling plant. The following elements are shown in the UI:

- Drink ingredients
- An animated bottle filling process
- The bottle labeling process
- Distribution (stock, maps, etc.)
- Metadata (users, alerts, etc.)

The application also required the following elements:

- Usability across multiple screens with adaptive design, including a 1024 x 600 touch screen, a 2560 x 1600 high-definition 10-inch tablet, and a 15-inch 1024 x 768 capacitive touch screen
- The ability to compensate for the very high pixel density on a high-definition tablet so that fonts and icons are easily readable for average users
- A user interface with a flat or minimalist design
- A slider panel for settings similar to an iPhone
- Different color design themes switchable at run time, for example a day and night design theme
- A UI with pop-overs and pin dialogs
- Icons for main menu items
- Fading animations when switching screens and changing the size of navigation bars
- Round animated progress controls

Development approach

Qt demo

A developer who was quite experienced in C++ and Qt but with little QML experience implemented the application in 160 hours. The developer was able get help from others with Qt and QML knowledge to get things done more efficiently.

HTML5 demo

Once the Qt demo was complete, the same developer developed the HTML5 version of the bottling demo. The developer had knowledge of HTML5, but was not experienced with AngularJS. 160 hours were also spent developing this demo.

Implementation details of the Qt version

The application uses QuickControls 2 with custom styling. The user interface is written in QML with models and logic implemented in C++ classes. Temperature and pie indicators use a mixed approach with graphics made with QPainter and wrapped into QML components.

Overflow animation of progress bars and pie indicators consists of two animations being run depending on value change:

- A simple NumberAnimation running if the next value is greater than the previous value
- A sequential animation consisting of two NumberAnimations otherwise:
  - NumberAnimation from previous value to maximum value
  - Unanimated reset of the current value to 0
  - NumberAnimation from 0 to the next value

Implementation details of the HTML5 version

There are different ways to handle complex animations in HTML applications:

- jQuery/JavaScript approach. Although it provides the best portability and enables complex animations easily, it was not used because it is outdated and very slow
- CSS3 animations:
  - transition: activates on property changes. Used in simple animations and corresponds to NumberAnimation in QML
  - animation with keyframes: can be used to implement complex animations

Overflow animation is not implemented in the HTML5 demo because there's no easy way to reproduce complex sequential animation from arbitrary value to arbitrary value:

- CSS3 transitions: it was necessary to animate from the previous value to the max, then from 0 to the next value. This would require at least two separate animations that must be watched to detect their start and end. Another challenge was that the value is changed but the animation is incomplete
- CSS3 keyframes: the nature of keyframes requires specifying static to/from values to animate. This was not possible within the scope because the values are dynamic and there are at least 100 * 100 combinations of values (for percentage indicators)

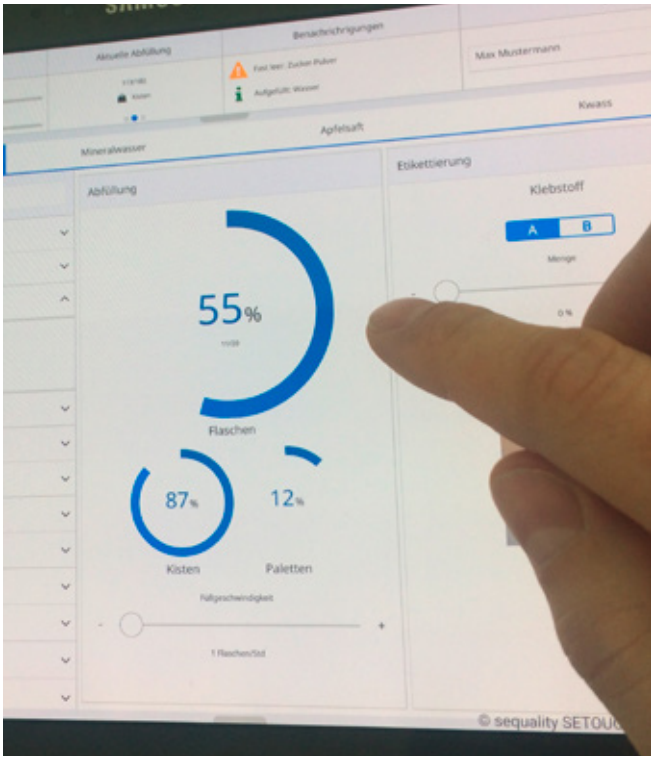
Hardware setup (both versions)

- Raspberry Pi 3
  - 1.2GHz 64-bit quad-core ARMv8 CPU
  - 1 GB RAM (50% shared with GPU)
  - VideoCore IV 3D graphics core
- 15 inch capacitive Touch-Display, 1024 x 768

Software setup (both versions)

- 32-bit Raspbian Jessie (March 2017)
- Qt/QML Demo: Qt 5.7.0, the "vanilla" version, cross-compiled for Raspberry Pi 3 without custom patches
- HTML5 Demo: stock Chromium browser from Raspbian repositories

Results after 160 hours of development



View a video comparing the Qt QML and HTML5 demos at https://vimeo.com/207307640

Note: neither demo is a fully featured application. For example, some buttons don't work and with the HTML5 demo there are bigger issues concerning browser compatibility and touch gestures. However, these demos clearly show the strengths and weaknesses of the technologies.

Features comparison

	Qt	HTML5
Dynamic theme switching	x	x
Lists	x	x
Table view	x	x
Dynamic search	x	x
Swipe gestures	x	x
Map	x	
Virtual keyboard	x	

Comparison of the Qt QML and HTML5 demos

Performance

- The Qt QML UI was faster than the HTML5 UI with our setup. The performance difference is likely due to the lack of proper OpenGL support by the Chromium browser on Raspberry Pi, whereas the Qt QML demo is rendered using OpenGL by the nature of Qt Quick Scene Graph
- Usage of AngularJS didn't affect performance in the HTML5 demo since data binding or DOM generation is not used extensively. The whole DOM tree is generated once at application start, and there is a small number of models watched by Angular engine
- Enabling GPU rendering on Chromium in chrome://gpu doesn't fix the HTML5 demo's performance problem. In fact, the CPU is utilized even more, which leads to overheating

Power consumption

- The Qt QML UI used less power than the HTML5 demo – HTML5 is rendered on the CPU only and therefore CPU utilization remains high

Browser engine

- Browser Engine Blink was the layout and rendering engine used by the Chromium browser on Raspberry Pi, Chrome browser on Android, and Chrome browser on Windows Desktop and various Linux flavors. The HTML5 demo was tested on Chromium for Raspberry Pi, Chrome on Android, and Chrome on Windows, and the touch event processing varied on these three platforms. For example, the slider widget didn't work as expected on Raspberry Pi, but the same code works well on Android and Windows. Moreover, Chrome on iOS uses WebKit as the layout engine due to Apple Store's limitations, meaning additional testing effort is required when targeting this platform
- Both demos use V8 as the JavaScript engine

Styling facilities

Qt QML approach

- QML applications do not offer any decoupling of styling, layouts, and actual components
- The property binding mechanism helps to implement model view-based architectures
- Colors, font faces, and styles must be specified explicitly for each component without any property inheritance
- Sizes are specified in pixels only, so implementing an application that should run on devices with different DPIs might be tricky, the property binding mechanism is used to bind QML element sizes to available pixel space
- Dynamic theming can be achieved using a singleton object referenced by all other components. In this case property binding helps to achieve dynamic colors, font sizes, and faces
- Dynamic layout requires either instantiating several components/layouts or using components in Loader. With a high degree of responsiveness in the requirements, the application structure might quickly become confusing
- Code reuse might be achieved by nesting QML objects or using C++ classes and class inheritance

HTML5/CSS3 approach

- HTML applications can be easily styled with CSS
- Very good decoupling of styling and markup is possible
- Support for CSS breakpoints and usage of Flex layout enables the creation of fluid and responsive user interfaces with little overhead
- Important property inheritance (font face, size) works automatically
- Sizes can be specified in relative units, which helps a lot in displaying UIs on devices with different DPIs
- Dynamic theming or a totally different layout within the same screen size can be achieved by loading another CSS file

- Usage of external CSS authoring tools like Less or Sass provide for a high degree of code reuse like variables, functions, and mixins
- CSS authoring tools usually contain functions to convert and manipulate color models, so it's easy to create different color schemes

Features comparison

	Qt QML	HTML5
Decoupling of styling and components		x
Dynamic layout of components	x	x
Styling property inheritance		x
Dynamic font faces	x	x
Relative units for sizes		x
Code reuse	x	x
Manipulating color models	x	x

Project structure and modularization

An important aspect of any project is keeping it structured, comprehensible, and modularized. Project parts should be decoupled as much as possible so that a change in one module causes as few problems as possible to other modules.

Some frameworks, like AngularJS, offer a project structure model from the beginning. Some mature programming languages like C++ have project structure patterns that are used by most programmers.

Defining module interfaces in any way makes it easier to wire project parts together. Singleton entities are required in most data processing applications. Although they might not be technically implemented (as a singleton is considered anti-pattern) data models are a good example of a semantic singleton entity.

Given the dependencies – for example, the fact that Users must be instantiated before Theming because Theme depends on User – the task of maintaining a singleton in an application might become tricky.

C++/Qt/QML approach

- Implementing logic in C++ and keeping views in QML is a way of naturally modularizing a project
- C++ programmers tend to put C++ classes into two files: a header and an implementation
- QML still doesn't have any common patterns for modularization adopted by most developers, so it's not uncommon to see different approaches from project to project
- The interface of a component is defined by its declared properties, slots, or Q_INVOKABLE methods
- There is no concept of property visibility in QML components per se, all exposed non-read-only properties can be overwritten by component users
- QML offers many choices on code recycling: inline components, components in separate files in the project structure, libraries in import paths (can also be manipulated from C++ code), or creating components dynamically from a string
- Singleton entities can be exposed to QML in three ways: registering a singleton type, setting a context property, or declaring a QML component singleton in qmldir
- All of them require special treatment if the object being instantiated has dependencies from other QML or C++ objects, i.e. how to ensure in C++ code that another QML singleton is instantiated
- Developers tend to put all context properties and type registrations to main.cpp or to plugin.cpp which leads to enormous wiring of these files with dependencies and prevents good decoupling of otherwise independent modules
- Making a QML component singleton requires putting it into a library
- QML nests scope by default and it can't be turned off – misuse leads to loss of readability very quickly
- Keeping QML parts of the project neat and maintaining readability requires good discipline from a programmer
- Working with actual data doesn't require any extra servers or applications, Qt offers networking, file I/O, database drivers, etc

HTML5/CSS3/AngularJS approach

Note: AngularJS was chosen to implement this demo. Other frameworks might require other techniques.

- With its directives and components, Angular allows the introduction of new tags to HTML markup, effectively incapsulating implementation details
- Directives and components require an HTML template and a controller (a controller is optional for a directive)
- Controllers, directives, components, and HTML templates are usually stored separately in separate files

- The interface of a component or directive is defined by its bindings. Bindings also declare data flow (two-way or one-way in either direction)
- Styling done in CSS is completely decoupled from all other parts of the code, although maintaining good readability and structure requires effort and external tools
- Angular offers a convenient dependency injection mechanism
- Angular offers a single mechanism for singleton entities
- Angular has a concept of scopes that optionally allows the capture of data from parent scopes, though misuse easily leads to loss of readability
- A special topic is managing separate component/directive/template files in AngularJS projects:
  - in the most basic scenario, all these files must be manually included in the index.html, which leads to a radical increase in loading time
  - in the more common scenario, these files are processed by an external tool to produce a single output file which is included in the index.html
  - Actual data processing (working with a database, file I/O, working with sensors) would require an HTTP- or a WebSocket-Server that processes upon request from the web interface. This introduces additional complexity to the project

Features comparison

	C++, Qt, or QML	HTML5, CSS3, or AngularJS
Well-defined initial project structure	x	x
Code recycling using components	x	x
Managing component dependencies	x	x
Effort to maintain project structure	x	x
Working with data directly	x	

Testing and debugging

Qt/QML

QML applications use a rendering engine provided with Qt. Normally the Qt version required can be shipped with the application, which guarantees a consistent environment for the application.

HTML5/AngularJS

Generally, AngularJS applications consist of many JavaScript sources that are concatenated together (possibly with dependency resolution) and loaded as a single bundle by the browser. Usage of transpilers and polyfills (for example, writing in ES6 and transpiling it to ES5 in the end product) increases debugging effort as the produced code is partly not the same as the actual one. Using source maps helps but they have to be correctly placed and recognized by the browser. Typos or errors in AngularJS code are often silently ignored and have to be investigated very thoroughly. There is quite a steep learning curve with AngularJS when it comes to data bindings and manipulating DOM in components, leading to increased development efforts for quite simple interactions like swipes or drags.

AngularJS has encouraged test-driven development from the beginning (see Google's tutorials on AngularJS). QML and Qt provide unit-testing frameworks as well, although do not push into using them. Overall, a TDD approach might be extremely useful for end-product quality but requires a lot of effort in the creation and maintenance of tests with good coverage. This effort often not only compares to, but also exceeds the effort of writing the actual code that is being tested.

The fact that HTML5 applications can be executed on a number of platforms – and a number of browser engines on each platform – multiplies the testing time correspondingly. Requiring a specific version of a browser might not be an option for tablets or smartphones where updates are installed automatically. Even on desktop computers, the default setting is nowadays to install browser updates without any interference from the user.

Sustainability of the technology

Web applications

With the rapid development of web applications in recent years, new client-side and server-side frameworks are constantly emerging. Major new versions of frameworks are released in short cycles and adopt new patterns in web engineering. This might discard backwards compatibility and support for older versions, increasing maintenance costs. Projects with a longer lifespan may use outdated frameworks and libraries until it becomes cheaper to write everything from scratch using completely new technology and architecture.

Moreover, the JavaScript language itself has undergone revision in recent years. It now incorporates features like scoped variables, classes, and modules (in ECMAScript 6), leaving web developers to deal with lack of support for all the features of the language in major browsers. Supersets of ES6 like TypeScript are actively used as well, and are not directly supported by browsers, forcing web engineers to use transpilation to the most portable old version of JavaScript. Luckily, the new versions of the language are supersets of the older versions and maintain backwards compatibility.

Apart from using existing frameworks, there's always a temptation to make your own in-house solution to exactly fulfill the requirements of the project. This might be considered feasible as long as the developers working on it don't leave the project, otherwise it would be yet another framework to learn with most probably non-existent or minimal documentation and no internet resources – creating the need to rewrite the application using existing frameworks or inventing a new one.

The nature of web engineering forces developers to use a complex multi-step build process to produce a working application. Not only do frameworks change, but the build tools change rapidly as well – like Grunt, Gulp, webpack, etc.

Another question to consider is the browser on the client side. With multiple rendering engines on the market, the application has to be tested for all of them and on all platforms. With extremely short release cycles of browsers (like Firefox) and seamless updates (like Chrome or Edge), it's virtually impossible to keep up. Targeting older browser versions also brings problems: they don't support newer CSS/JavaScript features and might behave differently when rendering. A supported version bump in the future doesn't help if some users keep the older version of their browsers and are therefore unable to use the application anymore.

The fine granularity of popular frameworks forces developers to use several frameworks in one project to achieve their goal. Each framework or library has its own style, integration level, documentation, release cycle, and level of support.

Though the web application stack will somehow stabilize in the next few years, it's important to make good decisions on what technologies to use, so that the project won't have to be rewritten from scratch multiple times.

Sustainability of C++/Qt/QML

C++ is a mature language that has proven to be conservative and slow changing. The adoption of the major C++11 standard in projects followed by its C++14 extension is fairly slow because of a lack of support by major compilers. The 4th edition of Stroustrup's The C++ Programming Language including C++11 features was published in 2013, two years after the standard was ratified. All the C++ standards up to and including C++17 are backwards compatible with the older version.

The Qt framework has been on the market since 1995, with major versions released in 1999 (Qt 2, lifespan of 2 years), 2001 (Qt 3, lifespan of 4 years), 2005 (Qt 4, lifespan of 7 years) and 2012 (Qt 5, lifespan of at least 5 years). Source compatibility was largely maintained during the transition from Qt 4 to 5, whereas previous transitions broke the compilation process. That is, the transition should have been made either completely or not at all. Qt 4 introduced Qt3-support classes for a smoother transition to the newer version of the library.

Paradigms and approaches used in Qt have remained mostly the same throughout recent major releases. That means having learned Qt once, one is able to develop Qt applications for at least the current and next major release. With a median cycle of major releases every 4 years, this typically means sustainable project development for at least 8 years.

As for now, Qt 5.x is developed in a 6-month cycle with API and ABI compatibility.

The Qt framework covers most areas of application development one might need in a project offering uniform class names, a single paradigm throughout the whole framework, accurate documentation, and good portability.

QML appeared in 2010 in Qt 4.7 and has been developed and polished constantly ever since. There's been evolutionary development of the language from the beginning while maintaining backwards compatibility. Moreover, the module version system ensures the correct outcome of an application being processed by newer versions of QML engines.

As mentioned earlier, unlike any HTML5 framework or library, Qt is a general-purpose framework and more or less self-contained – an application using Qt wouldn't require additional libraries for widgets, networking, databases, serial bus, etc.

Summary: comparing Qt QML with HTML5

Performance

Overall

- Both technologies are capable of separating design from business logic
- Both technologies are capable of rendering pixel-perfect UI designs on multiple platforms

Speed

- The QML version of the demo is faster than the HTML5 version
- With HTML5, latency time between user interaction and user interface response is higher with our setup

Features

- The QML version has complex swipe gestures (setup-slider, multi-touch zooming) that are not available in HTML5 out of the box
- The QML version has a virtual keyboard – no such component is available with HTML5 out of the box

Platform

- Qt-based applications are compiled for the target, meaning that in terms of user observation, they behave exactly the same no matter which platform they run on
- HTML5-based applications run on the browser of the target (e.g. Chrome), meaning different platforms can show different behavior as the browser might use different rendering engines depending on the platform. Browser testing is a major development cost factor in HTML5 based applications

Connection to hardware

- Low-latency sensor connections to direct hardware components are easier with C++/Qt as you have direct contact to the embedded hardware
- With HTML5, if the application has to react quickly to sensor input (for example, CAN-Bus-Value, GPIO-Pin, RS232, bluetooth, gyrometer), it can get very difficult to route this signal through the http-server, browser, and javascript to the HTML5 frontend

Animations

- Qt QML has built-in support for animations in the language, and transitions can be fine tuned
- With HTML5, there are several ways to implement animations, which can make it quite tricky to get exactly what you want from an animation
- Modern embedded applications benefit by having a dedicated OpenGL-capable graphics processer that can render animations in a smooth way. Making use of the OpenGL processer in an HTML5-browser-based setting is somewhat tricky – whether actual rendering can be done with the help of OpenGL depends on the browser. In the HTML5 demo with our setup, most rendering is done only on the CPU. In the QML demo, all rendering is done by the OpenGL-based engine

Testing and debugging

- Both HTML5 and QML can get somewhat tricky to test, which is true for most complex interactive visualizations
- With QML, complex logic is usually implemented in conventional C++ code, which offers the benefit of a strictly typed programming language - an advantage when testing and debugging applications
- With AngluarJS, all logic is written in a weakly typed environment, unless you use TypeScript, which is again a technology that has to be learned and used
- Generally speaking, one uses third-party tools (e.g. Babel, require.JS, webpack) to generate the runnable HTML5/AngularJS application out of a project. As a result, the code running in the browser is often not the same as the code written in your IDE. In our opinion, writing applications in AngularJS requires a more profound knowledge of the whole technology stack to develop and debug complex applications. Additionally, the technology-stack-components within an AngularJS application change rather rapidly, which makes it even harder to follow and understand

Sustainability considerations

- Qt has been around since the 1990s and Qt QML since 2011
- Modern HTML5-based applications that use frameworks like AngularJS are relatively new and undergo changes from year to year – a valid question is whether AngularJS (or any other currently trendy Javascript-library) will still be a relevant HTML5-technology in 10 years

About Sequality

Sequality software engineering is an Austrian software consulting company that is your partner for creating solutions in the area of industrial applications, touch display user interfaces, and embedded software applications.

Usability plays an important role when creating our applications. Together with usability engineers and UX designers, we can ship leading-edge UI technology applications that contain user-friendly functionality, look great, and deliver a seamless user interface experience.

sequality software engineering e.U. • Softwarepark 26A-4232 Hagenberg
www.sequality.at • t. +43 7236 26 101 • m. +43 676 97 72 681
stefan.larndorfer@sequality.at • office@sequality.at