sequality
SOFTWARE
ENGINEERING

HTML5 vs. Qt –
Full Software Stacks in Comparison

sequality software engineering
Softwarepark 26, A-4232 Hagenberg
www.sequality.at

Executive summary

HTML5 and Qt are both terrific technologies, and choosing which one to use in your next project can be challenging. Because comparing Qt, a full-stack development framework, and HTML5, a standard for a modern implementation of an HTML application in a browser, is like comparing apples and oranges.

This white paper compares an HTML5 application, including its backend software layers and the architecture of a full-stack Qt application. We identify the differences in software architecture and how they impact both your application and your entire product strategy. We also account for the extended ecosystem, such as what third-party tooling and developer communities contribute, and how your choice of technology defines your choice of potential target hardware.

This white paper doesn't aim to pick a winner, but to increase your awareness of which technology better fits a given situation, when it makes sense to combine the two, and how to make sure you plan for an open, scalable architecture. The focus of this white paper is on embedded devices and industry display panels. While a lot of the basics apply to a desktop or mobile environment, some of the nuances are more relevant for the embedded and industrial fields.

> "There is no silver bullet; neither Qt nor HTML5 is strictly better than the other in every context. You should base your technology choice on the overall architecture, the full stack."

Table of contents

HTML5 vs. Qt – an Incomplete Comparison

Go Web or go native? In the debate which of the two creates the best HMI, the backend often takes a back seat. The frontend technology (e.g., HTML5 or Qt Quick/QML) is, of course, a crucial part of any control software – how else would you visualize it? However, without the backend, which sends data to the control system, integrates serial interfaces such as the CAN bus, or stores data in a database, the HMI would remain just a pretty face.

Qt provides a “full-stack offer”, while HTML5 is a standard for a modern implementation of an HTML application in a browser. As such, you need to combine HTML5 with components like a browser, a web server, and an engine to enable additional features, which means they also have to be part of our comparison.

Architecture Comparison: Functional Requirements

Before you design the software architecture, you should clarify what functionalities the software should have. To keep it simple, let us assume two basic functionalities, which are widely used in industrial, medical or automotive technology to display sensor data and to provide the user with information on a machine’s behavior and condition:

- displaying a GUI on an embedded Linux device with a touch screen
- processing UI input and exchange data with the hardware interfaces (CAN bus, Ethernet, GPIOs, etc.)

There are two main ways to interact with a device: Directly via built-in touch display or remotely via smartphone, tablet, or PC. All the important functions have to be accessible without an active Internet connection, which means the device has to be able to process and visualize data locally, as well as to connect to hardware directly. Some examples are control displays built into industrial machines, in-vehicle infotainment systems, or medical devices used for monitoring patients.

Comparing Typical HTML and Qt Architectures

Figures 1 and 2 show the most basic Linux-based application architecture with a backend and a frontend for HTML5 and Qt, respectively.

An HTML5 architecture consists of two essential parts: The web server and engine as well as the HTML5 application, which is executed within the web browser. While the HTML5 application takes care of the presentation layer, the web server and engine implement the business logic, handle local data and connect to hardware interfaces (e.g., CAN bus) via the standardized access layers provided by the Linux operating system (OS).

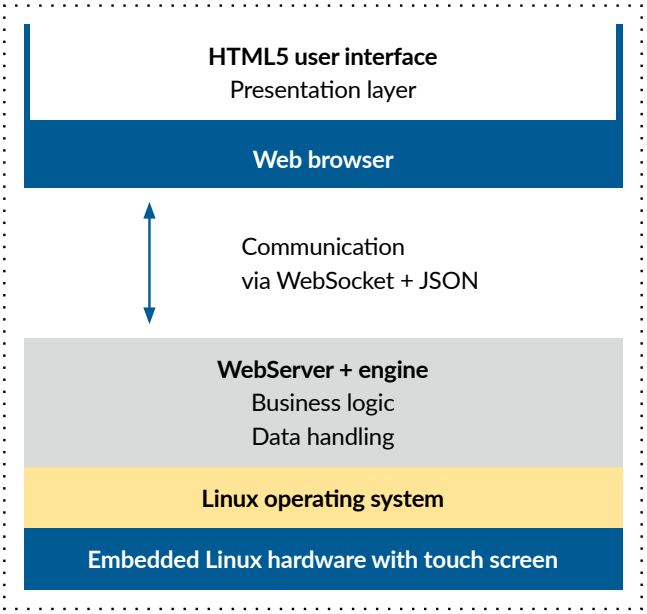

Fig 1: Basic architecture of an HTML5 web application

The HTML 5 Software Architecture

The colors in the architecture diagram represent the different technologies.

HTML5 User Interface: The presentation layer uses Web technology. Figure 1 refers to "HTML5 + JavaScript" for brevity's sake, but in reality, this layer would likely include a variety of other technologies. A modern implementation would most likely take a single-page-application-framework approach, such as Angular or React, which can add functionalities via packages.

Web browser: The browser on the device executes the web application. Most popular browsers today are written in C++ and available also as open-source packages for embedded Linux build systems (e.g., Yocto or Buildroot). Usually, you do not want to modify the source-code of well-proven existing browsers as this would mean that you would also have to maintain complex browser source code packages on your own, which would put a significant strain on your development resources.

Web server + Engine: The Web server delivers the web application as HTML files to the browser. We also need an environment that provides and forwards data to the Web application. Either the web server environment can do this on its own (e.g., node.js has some modules that enable CAN bus data access via JavaScript), or via a separate server process (= engine) in C ++, Java or another language, that sends the data via Web-compatible communication protocol (e.g., a web socket with JSON telegrams).

This section can be considerably more complex in reality because if you use hardware interfaces or specialized communication protocols (e.g., OPC-UA), you will most likely need additional technology besides the web server.

Linux OS: All scenarios presented here assume the use of an embedded Linux OS. This layer is generated via the build system and includes the drivers and software modules required to run the layers above.

Hardware: The OS runs on the hardware. Typically, it consists of a modern CPU (e.g., i.MX 6 dual-core), connections for CAN bus, Ethernet / WLAN, and a touch display.

To sum up, developers have to deal with at least four different technologies to create a web application on an embedded product: The HTML5 UI, the browser, and the web server and engine.

The Qt Software Architecture

The most basic Qt architecture for an embedded device (fig 2) consists of a single block.

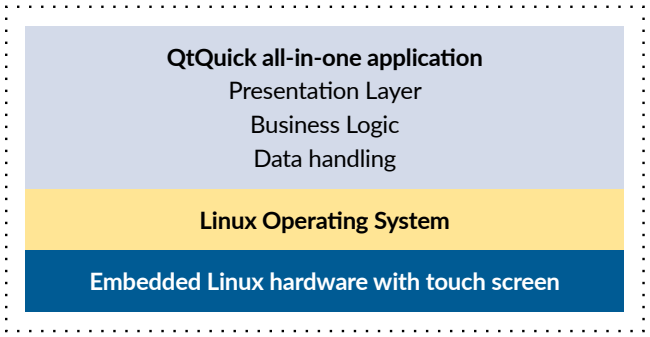

Figure 2: Basic architecture of a Qt Quick application

Qt Quick all-in-one Application: You would not likely use this architecture for more complex applications. However, if you pick the right software architecture, this approach can yield significant benefits for embedded devices. The more aspects and layers one application process can cover, the more you can reduce the application's memory overhead and footprint. Most of the time, the internal software architecture of this layer has a presentation layer (often implemented with Qt Quick and QML), and a business logic layer in C++.

To achieve scalability and flexibility, systems with Qt-based applications often have a similar architecture as HTML5-based systems (fig 3): The presentation layer is outsourced to a separate application (Qt Quick/QML) and communicates via a WebSocket interface with the business logic layer, which itself works in a separate process. There are many advantages to separating the presentation from the business logic at the process level:

- The GUI does not necessarily have to run on the same hardware as the engine.
- It's possible to have any number of GUI instances that connect to an engine.
- Ways to remotely control the system via additional applications (e.g., QtWebAssembly) are easy to implement.

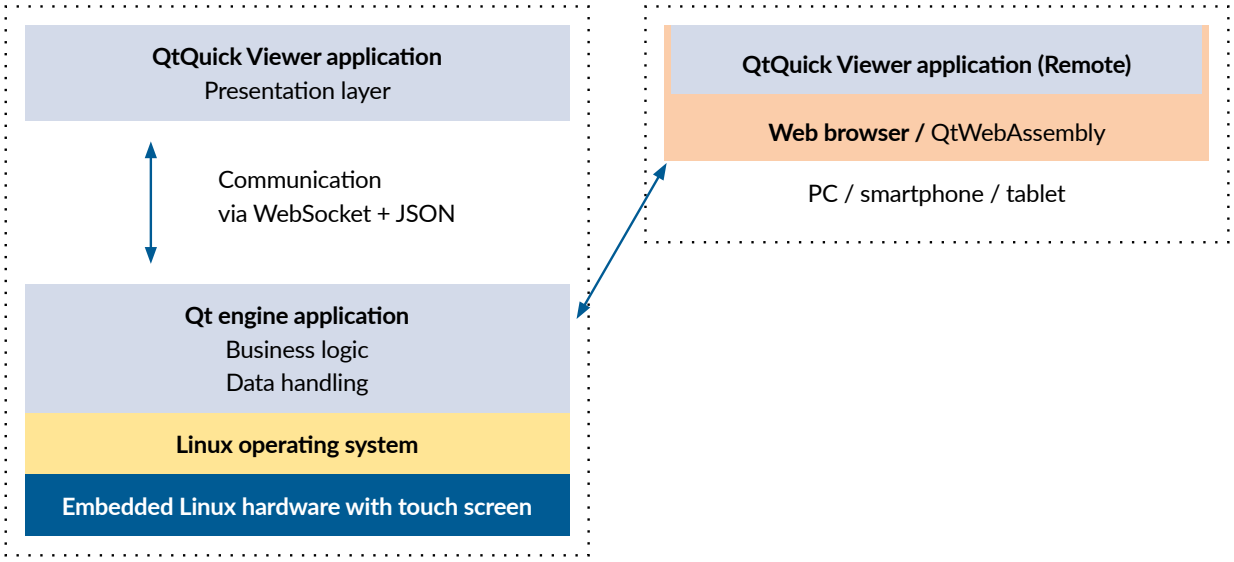

Figure 3: Architecture of a scalable Qt Quick application

QtQuick Viewer Application: This layer includes the presentation layer (QtQuick/QML) and a few additional C++ classes. Those ensure the integrity of the application structure and communication with the business logic layer (e.g., via WebSocket and JSON).

Qt Engine Application: This includes the business logic and data access layer. Access to hardware happens here exclusively. In most cases, this layer is implemented with Qt's C++ classes and benefits from many great hardware interfaces provided by the Qt Framework. The Qt engine application in fig 3 corresponds to the webserver and engine in fig 1.

QtQuick Viewer Application (Remote): Since Qt 5.12, you can deliver QtQuick applications via QtWebAssembly and execute them in a web browser, which serves as a virtual machine. This is useful for cases where you control a machine both locally (e.g., via touch screen) and remotely. By separating Qt Quick Viewer from the Qt Engine Application, you can use the same code for the Remote Viewer application.

The web browser below the Remote Qt Quick Viewer application does not belong to the software stack of the embedded device and is usually maintained by the PC, smartphone, or tablet user as part of system maintenance. So, apart from the Linux OS, only Qt is used in the software architecture, which streamlines maintenance and the identification of dependencies.

Being able to use the same framework for everything from the presentation layer to business logic and data processing is a significant advantage. The Qt framework even covers standard hardware interfaces such as CAN bus, Modbus, Ethernet, Bluetooth, etc. as well as many useful communication protocols like MQTT, OPC-UA or parser for XML or JSON. Thus, the Qt Framework enables a homogeneous architecture, which lets you avoid dependencies to other frameworks .

Compare Apples to Apples: The Full-Stack Eco-System Comparison

As we have established, if we want meaningful results, we need to compare full stacks: Qt and the entirety of HTML5, the web browser, the Web server, and the engine. The next paragraphs describe some decisive factors that might give a good indication of what direction a system should take.

HTML5: Breakout from the Protected "Sandbox" of the Browser

The browser executes its HTML5 application in a secure environment (sandbox) and is part of the UI. However, there are often questions on how to break out of this secure sandbox to implement certain features:

1) Integration of haptic hardware buttons or rotary push controls
2) Energy-saving measures like reducing the display's brightness during inactivity
3) Software update procedures for the whole system via USB or over-the-air
4) Hardware-accelerated video decoding and playback
5) Smooth kinetic scrolling supported by OpenGL and the built-in graphics card

While points 1 -3 can potentially be provided via an implementation in the "engine" on the web server (with correspondingly fast notification to the HTML5 application), points 4 and 5 must be secured in the environment of the browser itself.

One way to make sure to be able to break out from the sandbox is the use of an application-specific browser for a display panel. This enables direct access to the hardware with a link to the HTML5 browser technology. One example of how to achieve this is the usage of the Qt component "QtWebView". The Qt Framework provides this fully customizable web browser component based on the Chromium engine that can be used on embedded devices as well as on large industry display panels. (also see section "HTML5 with QtWebView"). With this approach, you can achieve a sandbox breakout by using the well-organized interfaces of this Qt component instead of having to directly interfere with the big and complex source code of a browser.

HTML5: Developer Tools on the Rise

As the popularity of web frameworks increases, so does the attention on their development tools: IDEs, browser-based code generators, debugging tools, etc. These tools not only have generic functionality such as syntax highlighting and syntax error checking in HTML and JavaScript code, but also provide code analysis and completion, boilerplate generation for specific web frameworks, and technologies such as Angular, vue.js, or Node.js.

Modern browsers like Firefox and Chrome have their debugging tools built-in to perform debugging locally, if necessary. Alternatively, you could connect a "primary" development environment such as VS Code or WebStorm to the browser.

In summary, the web development tools have become as powerful as the "major" IDEs for statically typed languages (e.g., Visual Studio, Eclipse, IntelliJ IDEA).

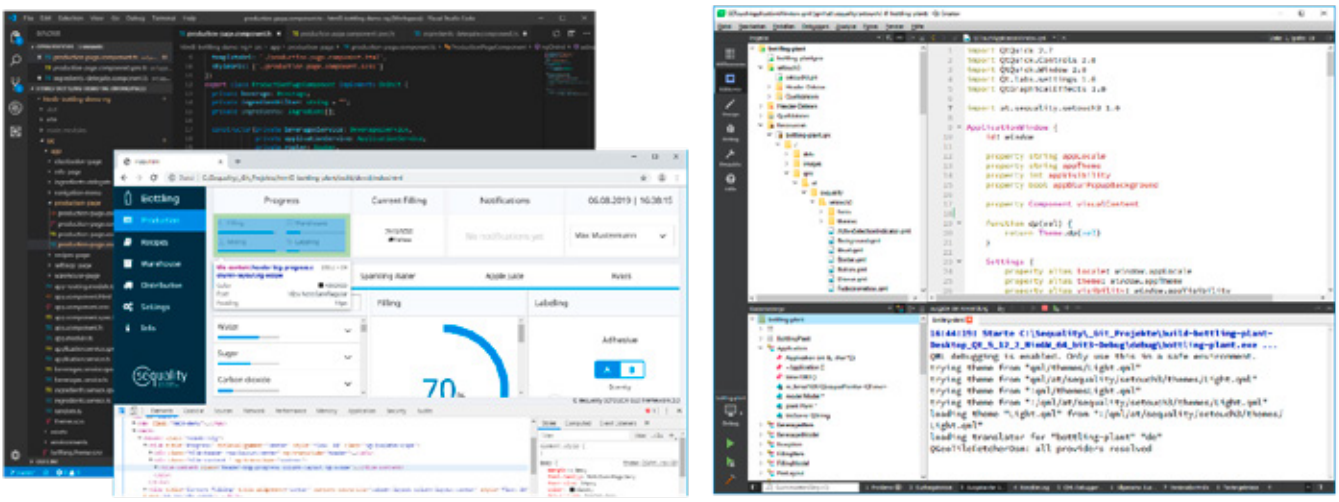

Figure 4: Left: An HTML5 development environment with Visual Studio Code in the background and DOM Inspector in Google Chrome. Right: The Qt development environment with visible QML code in the upper right corner and active output window below.

HTML5: Maintenance During the Product Life Cycle

Web applications on smartphones and PCs have a great advantage for software service providers: You do not have to worry about maintaining the application's runtime environment, i.e. the browser. Maintenance consists of regular browser and OS updates, which is "outsourced" to the end-user. If a particular browser version operates in a closed environment, the additional testing effort is low. If the testing environment is open, like with smartphones, tablets, or desktop computers, testing becomes an essential part of the software life cycle.

Besides the browser, you also need to add and maintain the HTML5 framework and its JavaScript libraries to the runtime environment. These components are being changed and updated frequently by the forces behind these libraries – most likely the big web players like Google (e.g., Angular), Facebook (e.g., React), and Microsoft (e.g., TypeScript). The innovation cycles of these big web players often do not harmonize well with the typical product life cycle of a machine or a device. Long-term support for specific software web libraries often is not covered. As a result, there is no guarantee that a JavaScript framework will be supported over the next 10 years.

Whichever web environment you choose, prepare to change the powerful HTML5 and JavaScript libraries or replace them with new ones over the next five years. Even within a web framework that has existed for years, incompatible changes can happen within a few months, rendering old versions almost obsolete, as the recent example of AngularJS and Angular show.

Qt: Maintenance During the Product Life Cycle

The Qt framework is also consistently maintained, improved, and developed by a big world-wide community. Again, a software maintenance strategy for your product is also essential to keep in mind. In contrast to the web eco-system, many parts of the framework (especially the C ++ classes) are stable since Qt 4 (release 2005). Unlike web technologies, Qt offers long-term supported (LTS) versions. Regular updates correct security holes or significant errors, so the code of the application does not need to be changed or converted. Updating the software for a new minor version (e.g., Qt 5.10 -> Qt 5.11) often doesn't require changes to the source code. Updates to a new major version (Qt 4 -> Qt 5) has so far worked without adjustments to the source code or was feasible with simple text substitutions. The planned update to Qt 6 is also supposed to remain source-code-compatible.

HTML5: Cross-Browser Testing and the Subtle Differences Between Browsers

Despite the improvements in HTML standards over the last 10 years, a web application still needs to be tested on different browsers. There are small differences between the three major browsers (Google Chrome, Mozilla Firefox, Apple Safari) that cause web applications to behave differently. While these differences do not affect applications on embedded devices in a notable way (because you only use a single, local browser), they become relevant when you operate a device remotely via smartphone, tablet, or PC.

Even if you only use one browser, there is no guarantee that the web application runs correctly after a version update. Testing with different browsers and browser versions will likely always be necessary in web architecture.

Control over Bits and Bytes

Complete control over bits and Bytes is not always necessary. That is why declarative scripting languages (see QML, JavaScript, HTML5) have become the default for writing GUIs. Still, especially in the embedded space, it is sometimes necessary to access bits and bytes to gain full control of the system. Qt lets you do this because a Qt Quick application can gain full control over the embedded system. In a system with web architecture, this is difficult, at least as far as visualization is concerned. If you want to achieve 3D acceleration, special hardware-accelerated effects, or optimize graphical transitions, you are restricted to the browser's predefined debugging options. That is unless you want to descend into the depths of complex C++ browser implementations.

C ++ – a Challenge for Young Developers?

No matter which technology best suits your requirements, you ultimately need developers that can work with the chosen technology, which requires a commitment to develop and train existing employees.

While this applies to both HTML5 and Qt, we have experienced that both young developers and decision-makers shun C++. There are many reasons for this: some older developers may still remember the 1990s when software was implemented with CORBA (Common Object Request Broker Architecture), Windows API, and MFC (Microsoft Foundation Class library) until they were "saved" by Java and C#. To the younger generation, who received their classic computer science degree in the late 2000s, C++ was only taught in an academic capacity or not at all. As a result, building a new team skilled in C++ can require a substantial amount of time and effort.

However, with Qt, the development of C++ software is simpler and more structured these days than with MFC or CORBA. You also do not need to know and use all of the language's modern and intricate features. The current C++ community is very active; the language is evolving, and because Linux is so popular within the embedded and server market, the importance of platform-agnostic software is increasing, which bodes well for cross-platform languages like C++.

The introduction of QML 10 years ago made it possible to create modern and fluidly animated graphical UIs. In contrast to C++, QML favors a simple syntax over strong typing. QML is optimized for building visual designs and offers an approachable entry point for HTML and JavaScript developers.

For an HTML5 architecture, there is an advantage in guaranteeing that the HTML5 UI must be developed exclusively with web technologies. All interfaces to the backend must be available via defined calls to the server. That is something you can also achieve with a Qt architecture (fig 2), but you will, even in the simplest Qt Quick application, come in touch with C++ to, e.g., implement a model-view concept.

Qt and HTML5: Typical Hardware Requirements

An earlier experiment comparing the same application implemented both with Qt / QtQuick and HTML5 (Angular) has shown that with the same hardware (Raspberry PI 3), the Qt Quick application reacted smoother and faster. We also observed that an incorrectly configured browser would lead to suboptimal cooperation between the OpenGL unit and the hardware. The CPU also had a much higher workload under the web architecture.

Typically, using a browser on an embedded device increases hardware requirements, as browsers and HTML technologies, unlike native applications, are not optimized for efficiency.

The following examples from the last years of our project experience give an idea of how typical hardware requirements have been implemented for both technologies:

- HTML5 application, Full HD industrial panel, Intel i3 2,2 GHz, 2GB RAM
- HTML5 application, 1024x768, Raspberry PI 2 (1.2 GHz quad-core), 1GB RAM
- Qt Quick application, 1280x1024 embedded panel, dual-core, 1GB RAM
- Qt Quick application, 800x600 embedded panel, i.MX 6 dual-core, 256MB RAM
- Qt C ++ application, 480 x 320 embedded panel, i.MX 28 single-core, 256MB RAM

Figure 5 shows the relationship between typical hardware requirements and cost per unit.

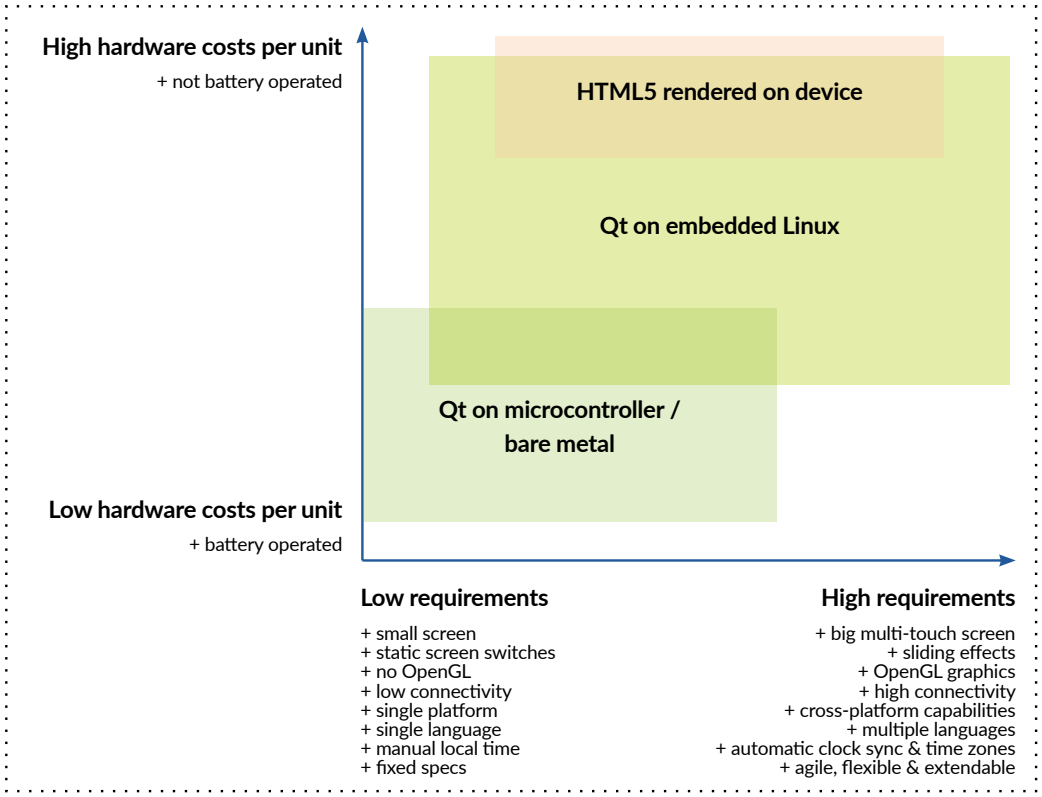

Figure 5: The "Embedded Visualization Map".

The higher the production volume and the lower the hardware costs of a device, the more critical an efficient software stack becomes. Since HTML5 technology typically has higher hardware requirements, it becomes less suitable for projects that aim for high production volumes with low hardware costs per unit. Other noteworthy dimensions in this graph would be the development costs and time to market, which are usually equally essential technology considerations.

Mixing HTML5 & Qt – The Best of Both Worlds

An increasing number of manufacturers have been combining a Qt Quick app on a touch display with an HTML5 application acting as the remote control. These are the advantages of such a setup:

- You can control the device remotely without having to install anything from an app store. This simplifies deployment and obviates the need to create and manage releases for different operating systems.
- Remote control is possible without an active internet connection if the HTML5 application is located on and delivered from the device.
- HTML5 applications are compatible with all major smartphones and browsers.
- A modern Single Page Application (SPA) framework as the basis of an HTML5 implementation (e.g., Angular), facilitates implementing a responsive browser app (see fig 5).
- Full graphics performance and responsiveness on embedded Linux display thanks to Qt Quick

Since the application on the device and the remote-control application in the browser usually do not have the same use cases, the UIs are not the same but tailored to their particular needs. The engine that contains the business logic (see fig 3) provides the same technology-neutral interface for both applications (e.g., JSON via WebSockets) and only needs to be developed once.

Before starting to develop a separate HTML5 application, one should also take a look at the latest version of the Qt technologies "Qt WebGL Streaming" and "Qt for WebAssembly". Both technologies can visualize an existing Qt application in a web browser: [1]

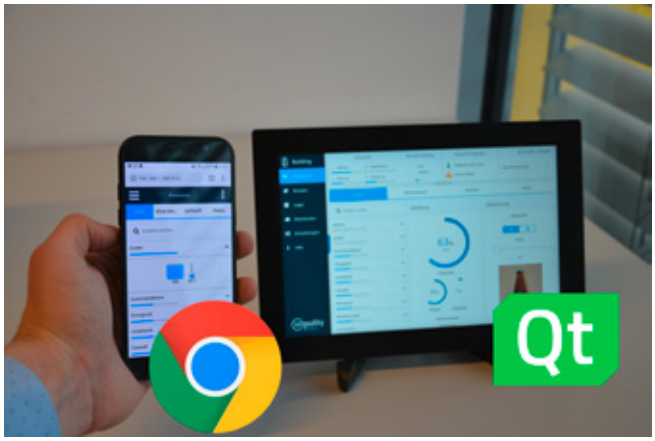

Figure 6: HTML5 application in a smartphone browser and Qt Quick application on an embedded Linux touch display.

- **Qt WebGL Streaming:** Similar to a VNC client, QML applications are streamed to the browser without the need for additional code to convert them into HTML. However, the application itself is still running on the embedded device. Note that this approach is also suited for a headless scenario. If you want several users to operate the application, you have to start several instances of the application, which should synchronize with each other. That way, you need to return to the split architecture with separate engine and viewer application.
- **Qt for WebAssembly:** WebAssembly is a kind of "virtual machine" within the browser that can directly execute a Qt application compiled specifically for WebAssembly. One should note that the architecture is split by definition (fig 3) because the WebAssembly application itself is not executed on the embedded device, but the user's browser. Data exchange with the engine requires an additional communication channel (such as WebSocket with JSON telegrams).

[1] https://www.sequality.at/en/know-how-2/cross-plattform-html5/

Software Stack Comparison

A complete system running an HTML5 or Qt Application consists of several software layers that run on top of the hardware layer. These software layers are mostly determined by the dependencies that are a direct result of the chosen architecture. You can set up these software layers in two ways: (1) Use an existing Linux distro and adapt it to your needs, or (2) use a Linux build system to generate the desired custom Linux system fully configured. Fig 7 shows the software layers of a custom Raspbian (1), while Fig 8 and 9 use a Yocto build system.

HTML5 Application with Google Chrome Browser "Off-The-Shelf"

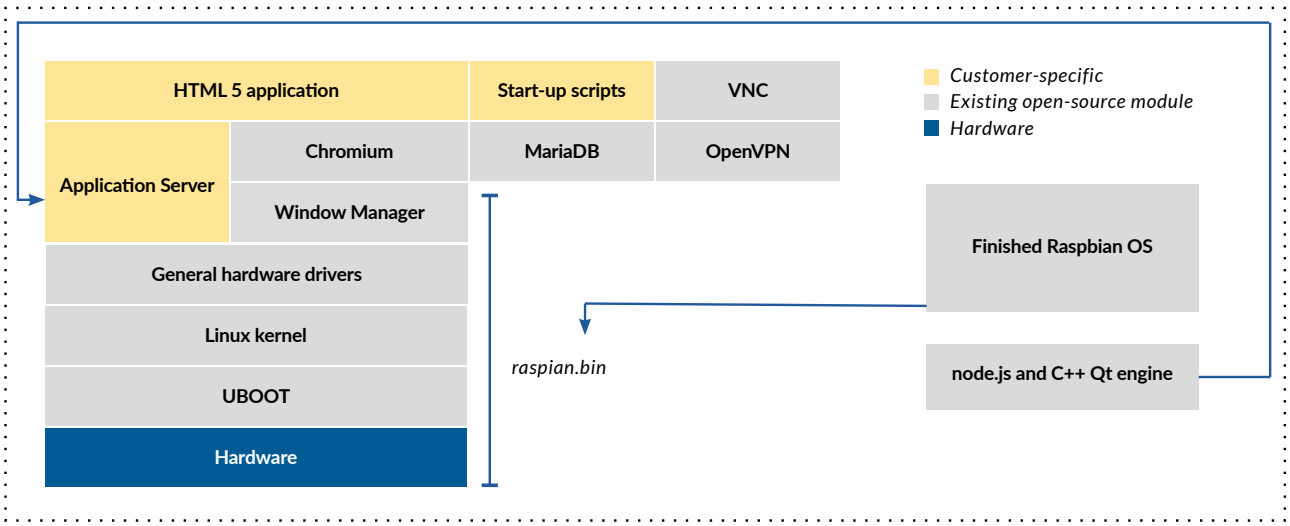

Figure 7: A standard Chromium browser with customized Raspbian operating system

Fig 7 represents the structure of our "sequality Bottling HTML5" demo application, which has a typical HTML5 Architecture (Fig 1). The light grey layers are available as ready-made open-source software. Customer-specific software modules are highlighted yellow.

Depending on the product, you may need to display several windows, or just a single main application, which is often the case. A single main application would not require a window manager. However, many standard browsers only run in an X server environment, which means that this particular software layer still needs to be included in the overall system.

How to Achieve a Secure Kiosk Mode?

The mandatory inclusion of the window manager comes with some potentially undesired functions. One example is "alt-tabbing", which swaps active windows, or, under certain circumstances, even closes a running application. Running Linux in "kiosk mode" can prevent this.

Qt Quick Application with Yocto

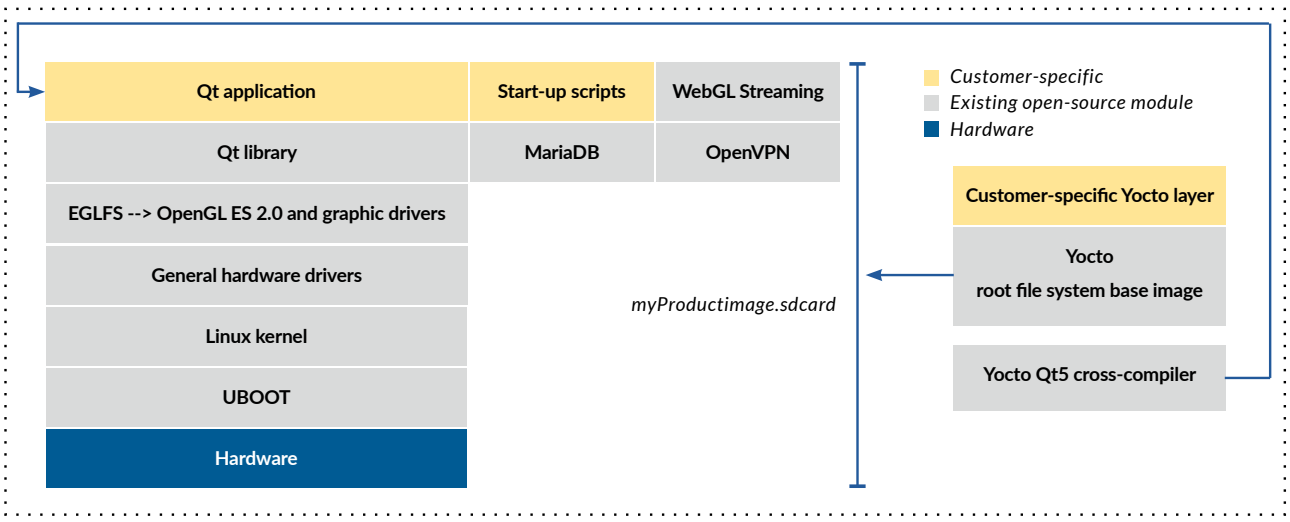

Figure 8: Software layer structure of a Qt application without a window manager

Qt applications can be operated directly via the OpenGL graphics interface, both with and without an X server (see EGLFS section here). Fig 8 shows a typical software layer structure of a Qt Quick application, as described in fig 2 or 3. In many cases, you can use the EGLFS interface to render the Qt application, instead of using a memory-consuming X Server. This approach will also enable a secure kiosk mode as well as faster boot times.

Using Yocto lets you repeatedly and accurately generate the product's entire root file system ("myProductImage. sdcard " in Fig8). Yocto also provides a Qt5 cross-compiler that can be installed on any Linux desktop environment and used to compile the Qt application for the target system.

HTML5 with QtWebView

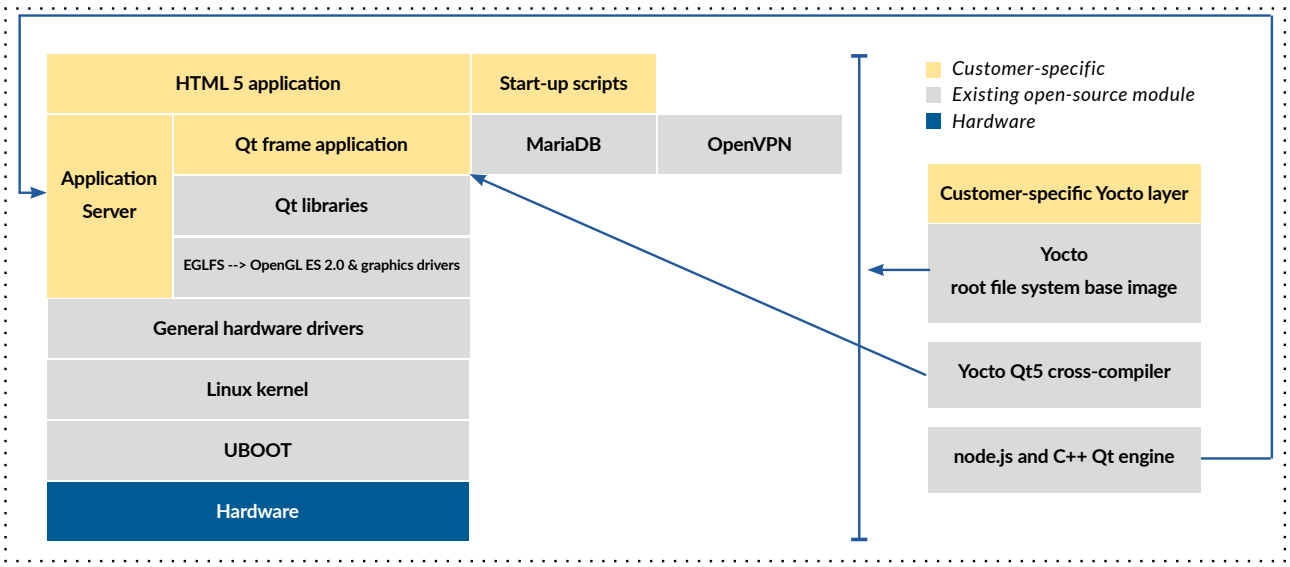

Figure 9: The software layer structure with QtWebView

If you decide to implement your software with HTML5, it makes sense to address the question of how to implement the optimal browser for its kiosk application, embedded device, or product. The standard browsers such as Google Chrome or Mozilla Firefox offer plugin interfaces, but no practical way to add custom extensions (or restrictions). The Qt Framework offers QtWebView, a Chromium-based browser component that can be tailored to the needs of a product.

The advantages:

- **No X server required:** Qt applications can use the EGLFS graphics interface to display a window. This keeps the software stack streamlined and protects it from security holes and "backdoors" via which the user could open additional windows or manipulate windows with keyboard commands. The use of kiosk software is not necessary.
- **Guaranteed OpenGL support:** The Qt Framework and the integration of the Qt Yocto Layer ensure the support of OpenGL.
- **Control over screensaver:** You may want to make your products more efficient by having them lower the display brightness after a period of inactivity, which can be solved with the Qt framework application around QtWebView with only a few lines of code.
- **Virtual keyboard integration:** Text input for an HTML5 application with a touch display requires a virtual keyboard. The Qt framework already has a ready-made virtual keyboard component that can be used with QtWebView. It is also possible to implement your custom virtual keyboard with Qt.
- The Qt framework application can directly connect to hardware switches or buttons on the device. That way, hardware events can be passed on both via a server as well as directly connected to the web browser software.
- **Stable release cycles:** The Qt Framework is developed in regular release cycles and is highly stable. This ensures that the built-in webview component is up-to-date for years to come.
- **Full control of the browser GUI:** There are certain GUI elements like the back or home buttons, loading bar, or mouse cursor you want to either avoid or load with features for security reasons. With standard browsers, this adjustment can be very tedious. With QtWebView, this and the control over the right mouse button and keyboard shortcuts is relatively easy.
- **Customer-specific error management and logging:** Should an error occur in the HTML5 application, it should be automatically logged for later analysis or even forwarded to a person in charge. This is difficult to implement with standard browsers. With a specific QtWebView framework application, you can, e.g., log endless loops in JavaScript and then visualize a customer-specific display. Also, in the case of a broken link or a failed connection to the server, meaningful screens can be served to the user.
- **Security and Safety:** By consistently reducing the browser to just the essential features of the product, the browser environment becomes more secure and less vulnerable to attacks.

Figure 10: The HTML5 sequality- Bottling demo in QtWebView on an embedded Linux terminal.

Qt Quick Application with Yocto and HTML5 Remote Control

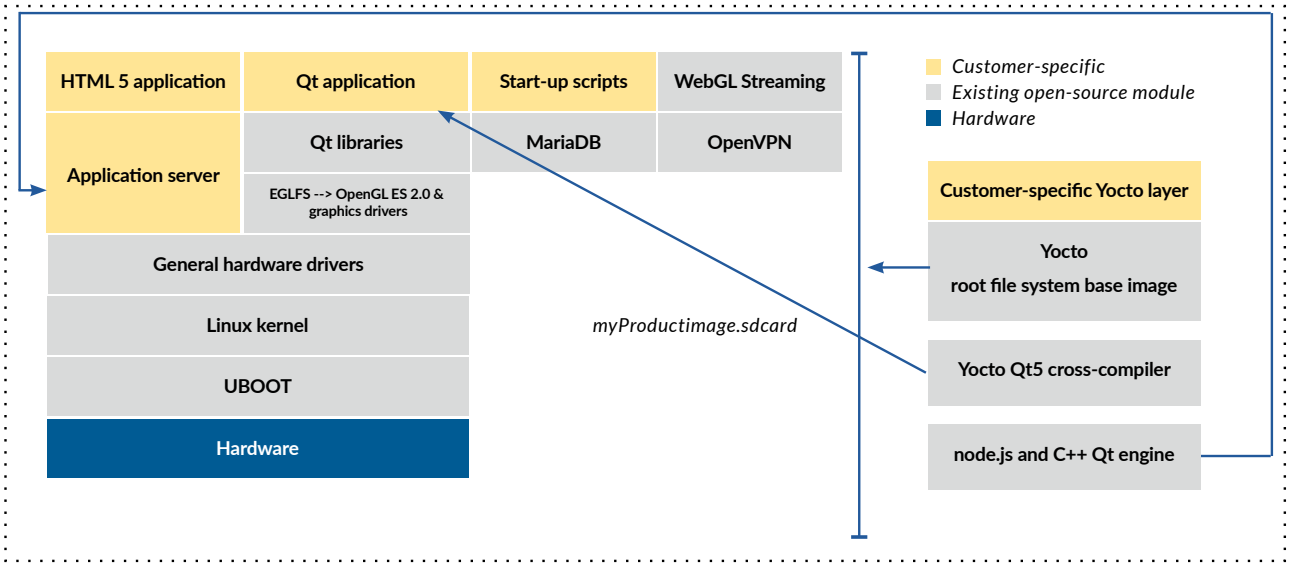

Figure 11: Software layer structure of a Qt application with HTML5 for remote control via smartphone

In some cases, it is interesting to use HTML5 to implement a remote-control application in addition to the native Qt application on the device. Typically, an external device such as a smartphone or a tablet serves as a remote control that uses the web browser on the smartphone/tablet to execute the HTML5 application on the device. In this case, the HTML5 application can only be accessed via an external device and its browser. Information on the device is only displayed via the Qt application. Both the Qt and the HTML5 application use the application server to share the same business logic.

Summary

Both HTML5 and Qt are terrific options to use for your product. There is no silver bullet; neither of them is strictly better than the other in every context. You should base your technology choice on the overall architecture, the full stack. Also, being aware of the dependencies between libraries and their stability is vital. Lastly, the stability and functionality of the entire software stack dictate whether a product can be used and expanded upon during its product life cycle.

If you go for an HTML5 approach, it is worth it to investigate specific modules within the Qt library, especially modules related to industry automation or backend protocols (Modbus, OPC-UA, etc.). Using Qt WebView represents the most straightforward implementation of a product-specific browser with stable update cycles and custom hardware support.

Whatever your decision: Make sure you plan for an open, scalable architecture. If you base your product on a scalable architecture with technology-neutral business-logic interfaces, your hybrid systems become a lot easier to implement. That way, you are free to employ the ideal technology in each situation, without having to implement any complex business logic twice.

References

Qt QML vs. HTML5 - a practical comparison
Stefan Larndorfer / Dmitriy Purgin, 2017,

https://resources.qt.io/whitepaper/white-paper-qt-vs-html5-1-practical-comparison
https://www.sequality.at/projects/html5-vs-qt-demo/
https://www.sequality.at/2017/07/25/html5-vs-qt-whitepaper-available/

Qt or HTML5? A Million Dollar Question
Burkhard Stubert , 2017,
https://resources.qt.io/whitepaper/white-paper-qt-vs-html5-2billion-dollar-question

Cross platform Qt & HTML5,
https://www.sequality.at/know-how/cross-plattform-html5/

Web HMI or native HMI - which concept is the better?
April 2019
https://www.elektroniknet.de/markt-technik/automation/web-hmi-oder-native-hmi-welches-konzept-ist-das-bessere-164639.html

Yocto project, software overview
https://www.yoctoproject.org/software-overview/

Buildroot
https://buildroot.org/

About the Author

Stefan Larndorfer is the CEO of sequality software engineering and manages various projects in the areas of automotive, industrial automation and medical engineering. Together with his team, he develops innovative software solutions based on embedded Linux. He is also a lecturer for C ++ Qt at the Hagenberg University of Applied Sciences and supports open-source development, as well as different research projects.

Stefan.Larndorfer@sequality.at

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

sequality SOFTWARE ENGINEERING