Migrating Applications from MPUs to MCUs

based on a Qt Quick unified architecture

This white paper shares our experience of porting an existing Qt Quick application to a microcontroller (MCU) with Qt for MCUs.

We chose to port our existing “HomeChef” application, which comes in a desktop, mobile, and embedded deployment. We used a unified architecture for the different deployments, which have different use cases and operating systems. All the deployments, including the MCU port, was done with one codebase and one toolchain to serve one brand identity.

We provide in-depth information about the overall process of shrinking a Qt Quick application down to 480kb with Qt for MCUs 1.0 and porting it to an STM 32F769. We explain how we could address the challenges in a surprisingly short timeframe while still achieving a performance typically associated with more powerful hardware.
## Contents

**HomeChef Eco-System: Qt Quick Unified Architecture** ................................................. 3  
**Development Challenges for Multi-Deployment Scenarios** ........................................... 3  
**Proving the Qt Technology** ......................................................................................... 3  
**Our experience with Design Studio and WebAssembly in the making of the HomeChef** ................................................. 4  

**Extending to Qt UltraLite** ............................................................................................. 5  
**Why did we want to develop HMI on the MCU?** ......................................................... 5  
**HomeChef Top-Down approach** .................................................................................... 5  
**Gap Analysis (what was not available on Qt for MCUs)** ............................................... 5  
**Challenges migrating to Qt Quick UltraLite (QUL)** ..................................................... 8  

**Overall Effort and Timing** ............................................................................................. 9  
**Conclusion/Learnings** .................................................................................................. 12
HomeChef Eco-System: Qt Quick Unified Architecture

Development Challenges for Multi-Deployment Scenarios
When your users want to enjoy their favourite software on multiple devices, they expect a consistent experience across the board. At the same time, your challenge to create a cohesive, unified product architecture increases, the more platforms you port your software to. Additionally, there is a shift towards incremental and iterative development processes that adds additional fragmentation to product creation.

In this fragmented context of product delivery, integration has become a key concern. To address this concern, we believe approaches such as the Unified Architecture Method (UAM) support product delivery across platforms by taking advantage of discrete user interfaces (UIs) and reusable components in cross-platform environments.

With increasing technical requirements, solutions that cater to new, innovative home equipment design become more complex and expensive. While easy-to-use MCUs underperform in these new sophisticated implementations, high-end industrial Microprocessing units (MPUs) come with higher hardware and software complexity, which can lead to high follow-on costs.

Proving the Qt Technology
To outline how to address all these challenges with Qt, we created a consumer electronics demo, “HomeChef”, as a real-life use case of multiple deployments as one ecosystem. The idea was that optimizing cooking (faster, cheaper, less waste, healthier) will be a key aspect of future sustainability. HomeChef keeps people updated with what they have in their kitchen and what they need to buy in the supermarket. We believe comparable scenarios will apply to other industries and use cases. HomeChef comes with three different applications and devices:

A desktop deployment to prepare your dishes of the week: Choose recipes from recipe books, favourites, or automatic recommendations, and add the ingredients to a shopping list. HomeChef can also collect nutritional information and show you statistics of how healthy you live. Instead of creating a native desktop application, we decided to deploy this desktop application to WebAssembly so that users can run this application without any installation, which makes HomeChef a showcase of Software-as-a-service (SaaS) made with Qt.

The second deployment was the mobile app, which has the same functionality as the desktop version but optimized to smaller form factors and made as a native App for iOS and Android.

Last but not least, we created an embedded cooking machine that gives step-by-step cooking instructions as you prepare your meals. We utilized a Garz&Fricke Santaro, a i.MX6Dual-based human-machine interface (HMI) panel running Embedded Linux with Yocto as its build system. Finally, as technology highlights, we added both WebGL streaming for remote viewing of a camera output and WebEngine integration for viewing Youtube streams.
All the deployments communicate via an MQTT\(^1\) broker in the cloud (or running locally for demonstration purposes).

Our experience with Design Studio and WebAssembly in the making of the HomeChef

We used Qt Design studio\(^2\) for the first time, and after initial training, it was easy to adopt in our development process. In the beginning, we used Qt Design Studio for prototyping design ideas, but soon it became an essential development tool.

---

1 MQTT is a machine-to-machine (M2M) protocol utilizing the publish-and-subscribe paradigm. Its purpose is to provide a channel with minimal communication overhead.

2 Qt Design Studio is a UI design and development environment for creating animated UIs and previewing them on the desktop or on Android and embedded Linux devices.
Extending to Qt UltraLite

Why did we want to develop HMIs on the MCU?
As an embedded HMI development company, Verolt was always interested in developing applications on embedded platforms like NXP and others. However, these applications almost always required Linux-based processors with GPUs and other hardware support.

We have always considered adding MCU-based HMI products to our portfolio. However, the vendor landscape so far has been quite scattered with incomplete offerings. When The Qt Company announced Qt for MCUs, a graphics framework and toolkit optimized for MCUs, it was an obvious choice for us to invest into this technology.

We will continue with an outline of our approach, the analysis we carried out for developing the HMI on Qt for MCUs, the gaps we identified, and the steps we took to overcome these gaps.

HomeChef Top-Down approach
Since the HomeChef application design and development on Linux were nearly complete, we decided on a top-down development approach for HomeChef on MCUs.

Divide and Conquer: The advantage of the unified architecture is that it lets you effortlessly break up a project into smaller, more manageable pieces.
We analysed HomeChef’s features and their compatibility with MCUs, had brainstorming meetings on feasibility, and regular live-coding sessions with validation. This helped the team overcome the unknowns and achieve the feature completion goal with minimal design changes.

Thanks to this top-down approach, we could
- Focus on the features most relevant to the customer
- Test each feature as soon as we finished it. Integration tests and user tests frequently happened frequently throughout the project, which made the outcome more predictable
- Send continuous builds regularly to the customer with every newly-implemented feature
Gap Analysis (what was not available on Qt for MCUs)
A microprocessor typically runs an operating system that allows multiprocessing and multithreading. The fact that microcontrollers do not run such an operating system would impact the application design. The table below shows a feature comparison between Qt for Embedded Linux and Qt for MCUs.

<table>
<thead>
<tr>
<th>Qt modules on Embedded Linux</th>
<th>Availability Qt for MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qt Network</td>
<td>Not available</td>
</tr>
<tr>
<td>Qt Quick Dialogs</td>
<td>Not available</td>
</tr>
<tr>
<td>Qt Quick Controls</td>
<td>With limitations</td>
</tr>
<tr>
<td>Qt Quick Layouts</td>
<td>Not available</td>
</tr>
<tr>
<td>Other modules (e.g. Qt MQTT, Qt Serialport etc.)</td>
<td>Not available</td>
</tr>
</tbody>
</table>

Developers need to be aware that Qt for MCUs does not support the Qt C++ modules. It does, however, let you use pre-existing code.

Figure 2: Deployment of HomeChef on an MCU
We decided to port the frontend, as the backend required more effort to integrate the MQTT and WIFI modules.

Qt for MCUs provides a subset of Qt Quick known as Qt Quick UltraLite\(^3\) (QUL), which addresses the resource constraints in MCUs. The trade-off to the QUL’s lower footprint comes in the shape of limited capabilities, compared to Qt Quick. To achieve the expected result, we had to simplify certain QML\(^4\) design elements. Below are some of the learnings during this development phase.

**Design Studio**

We developed the frontend with the help of Design Studio. Because there was no support for QUL at the time of development, some components that were dependent on Design Studio components and Qt Quick required rework.

<table>
<thead>
<tr>
<th>Design Studio component</th>
<th>Reworked for Qt for MCU</th>
<th>Reason for rework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circular progress bar</td>
<td>QML needed to be updated with timer-based instead of timeline-based animations</td>
<td>Design studio component Timeline is not available for Qt for MCUs</td>
</tr>
<tr>
<td>Charts</td>
<td>Completely redesigned the component in QML</td>
<td>QChart module is not part of Qt for MCUs</td>
</tr>
<tr>
<td>Cooking Steps</td>
<td>Minor modification</td>
<td>QML type support was not available</td>
</tr>
</tbody>
</table>

\(^3\) Graphics runtime optimised for high performance and low memory consumption on resource-constrained devices.

\(^4\) The Qt Modeling Language is a declarative Markup language for designing UI-centric applications.
Challenges migrating to Qt Quick UltraLite (QUL)
During the migration, we came across differences between QUL and Qt Quick. Here is a list of elements we needed to rework for HomeChef.

<table>
<thead>
<tr>
<th>Element</th>
<th>Qt Quick</th>
<th>QUL 1.0</th>
<th>Limitation</th>
<th>Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>QTimer</td>
<td>Available with limitations</td>
<td>Function Triggeronstart not available</td>
<td></td>
<td>Call explicitly start</td>
</tr>
<tr>
<td>Rectangle border</td>
<td>Not available</td>
<td></td>
<td></td>
<td>Overlapping Rectangle to show border</td>
</tr>
<tr>
<td>List View</td>
<td>Available with limitations</td>
<td>Horizontal scrolling not available.</td>
<td></td>
<td>Flickable property for horizontal scrolling.</td>
</tr>
<tr>
<td>Text Input</td>
<td>Not available</td>
<td></td>
<td></td>
<td>Custom type</td>
</tr>
<tr>
<td>Text Edit</td>
<td>Not available</td>
<td>-</td>
<td></td>
<td>Custom type</td>
</tr>
<tr>
<td>Property Animation</td>
<td>Available with limitations</td>
<td>Property animation to anchors is not available</td>
<td></td>
<td>Timer and Transform properties to Rotate the object. Prebake animation in photoshop</td>
</tr>
<tr>
<td>Java script functions</td>
<td>Not available</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Popup</td>
<td>Not available</td>
<td></td>
<td></td>
<td>Custom type with high Z-order</td>
</tr>
<tr>
<td>Font</td>
<td>Available with limitations</td>
<td>Font. Weight increases the memory footprint significantly. Font. Pixel size should be a fixed value</td>
<td></td>
<td>Dynamic resizing of font should be avoided.</td>
</tr>
<tr>
<td>Loader</td>
<td>Not available</td>
<td></td>
<td></td>
<td>Used visibility property for page loading</td>
</tr>
<tr>
<td>Scroll View</td>
<td>Not available</td>
<td></td>
<td></td>
<td>Flickable property for vertical scrolling</td>
</tr>
<tr>
<td>Drawer</td>
<td>Not available</td>
<td></td>
<td></td>
<td>Custom type with animation and visibility</td>
</tr>
<tr>
<td>Stack view</td>
<td>Not available</td>
<td></td>
<td></td>
<td>Custom type with visibility flag</td>
</tr>
<tr>
<td>Model</td>
<td>Available with limitations</td>
<td>Cannot declare in another file</td>
<td></td>
<td>Model and delegate should be declared in same file</td>
</tr>
<tr>
<td>Combobox</td>
<td>Not available</td>
<td></td>
<td></td>
<td>Custom types</td>
</tr>
<tr>
<td>Application window</td>
<td>Not available</td>
<td></td>
<td></td>
<td>No possible alternative, but planned for future version</td>
</tr>
<tr>
<td>ItemDelegate</td>
<td>Not available</td>
<td></td>
<td></td>
<td>No possible alternative, but planned for future version</td>
</tr>
<tr>
<td>Page</td>
<td>Not available</td>
<td></td>
<td></td>
<td>No possible alternative, but planned for future version</td>
</tr>
</tbody>
</table>
QUL Performance and MCU limitations

Individual animations were showing signs of performance degradation. To overcome this performance issue, we conducted several trials and finalised the solution by reducing computation and using a single timer for multiple micro-animations.

One such use case, a circular progress bar depicting the progress of kneading, was developed with design studio elements. In this particular case, we used a design studio element “Arc,” which was animated by “Timeline,” another Qt Design studio element. Since these elements were part of Qt Design Studio, we had to redesign the circular progress bar.

- The animation of the progress bar required more memory and power, so we had to take a timer-based rotation approach. We found that this approach significantly reduced the memory footprint and provided a smooth animation.
- At the end we ended up having a total application size of 480kB and being able to provide almost the same look and feel as the reference Embedded UI.

Overall Effort and Timing

The key to migrating any software to a microcontroller to deliver an embedded system with a viable software architecture at reduced development costs and a quicker time to market.

In the case of HomeChef, the choice of unified architecture and advantage of Qt’s offering on multiple platforms helped developers achieve a quick deployment on MCUs with a minimal learning curve. The lack of embedded software resources and development time required the reuse of software components. Here is an idea of the time and effort it took to port Qt Quick to QUL, based on our work on the four most complex screens in HomeChef for MCUs. We are basing our estimations on a developer of medium skill level with basic knowledge of QML.
Screen

Complexity: Medium
- This screen had to undergo a medium level of changes involving a horizontal scrolling list and other GUI elements.
- **Time required**: One day

Screen

Complexity: High
- Nested collapsible list: This component had to undergo GUI changes to make it work on MCU.
- To accommodate the small screen size, the UI had been changed with enlarged fonts and mouse areas.
- **Time required**: Three days
Screen

Complexity: Medium
- Animated circular progress bar
- Replacement of Qt Design Studio elements to timer-based animation
- Overlay of images has been used to display data and clip unwanted image data
- Time required: One day

Screen

Complexity: Medium
- Animated circular progress bar
- Rotation Animation
- Same timer used for multiple microanimations like fan or temperature.
- Time required: Two days
Conclusion/Learnings

We know from our customers that a common brand identity across all consumer/customer touchpoints is a crucial business driver. This applies not only to the consumer electronics industry like with our “HomeChef” demo but to many others.

With this demonstrator, we have been able to show that this is doable with one design and codebase across all deployment scenarios. As a result, fewer engineering skills were required, cooperation between design and development has been frictionless, the transition from prototyping to product development smooth, product iterations in sync, and components re-usable across the different UI flavours, versions, and platforms.

With Qt for MCUs, The Qt Company closes an essential gap in the low-cost-high-volume sector. As we have shown, the 1.0 version already comes with a comprehensive product-ready portfolio. Yes, things like improved QML type and SCXML, full Design Studio integration, or image processing (e.g. for 2.5D animations) would have been great to have, however, they haven’t been a show-stopper for us.

All-in-all, our proof of concept of the HomeChef ecosystem showed that it’s possible to reduce both the cost of engineering as well the bill of material for our customers with Qt for MCUs.

If not hampered by legacy code, a unified Architecture including MCUs is possible with Qt, and, done wisely, guarantees future innovation for engineering teams. As a service provider in this area, we are not aware of any better alternatives.

Need help? No Problem!
We can migrate to any thing. Please contact Qt to optimize your development projects.

Contact Qt at www.qt.io/contact-us/