Qt or HTML5?

A Million Dollar Question

Burkhard Stubert
Chief Engineer, EmbeddedUse

October 2, 2017

Burkhard Stubert is Solopreneur and Chief Engineer at Embedded Use with over 20 years of experience in software engineering. He has developed numerous embedded and desktop applications with Qt and QML. He was the first to give QML trainings back in early 2010, when QML was still far away from an alpha release. His latest major products include driver terminals for forage and sugar root harvesters, infotainment systems for US and European car OEMs, an in-flight entertainment system and a display computer for e-bikes.

He offers independent professional services for developing embedded systems – preferably with a QML GUI and Qt/C++ middleware. Burkhard worked and lived in India, England and Norway and moved back to his native country, Germany, a couple of years ago. In his spare time, he enjoys hiking, biking and skiing through the Bavarian Alpes.

エグゼクティブサマリー

本ホワイトペーパーは、世界有数の家電メーカーがWebテクノロジーではなくQtを活用してアプリ開発を行うことにより、いかにして数百万ドル規模のコスト削減に成功したかを考察／詳述するものです。
たとえばFacebookやNetflixは、自社名を冠したWebアプリを開発しました。しかし数百万ドルの開発費をつぎ込んだにもかかわらず、両者は4コアのARM Cortex-A9 SoCで、iPhone並みのユーザーエクスペリエンス（60fps・100ms以内のレスポンス）を提供できませんでした。
これに対し、インフォテインメントシステムや機内エンターテインメントシステム、収穫機管理端末、家電といったさまざまな製品が現在、シングルコアのCortex-A8 SoCでiPhone並みのユーザーエクスペリエンス（UX）を実現しています。
前述した世界有数の家電メーカーHAM Inc.（機密性保持のため、ここでは仮名を使用）は、WebとQtの両方でプロトタイプを構築して比較し、このようなUXの提供が可能であることを実証しました。

産業グレードのシングルコアCortex-A8内蔵NXP i.MX53 SoCを100万個単位で製造する場合、1個当たりのコストは約10ユーロです。Qtでの開発にはこの仕様で十分です。一方、Web開発向けに4コアのCortex-A9内蔵NXP i.MX6 SoCを同じく100万個製造するには、1個当たり約21ユーロかかります。

つまり Qtなら、ハードウェアコストを53％以上削減できるのです！

HAM社の場合、SoC製造コストの削減分はQt商用ライセンスコストで相殺されてしまうかもしれません。しかし合計コストで見れば、WebソリューションがQtソリューションを数百万ユーロ上回るのが実情です。しかもWebソリューションは高級品向けで、中級品や低価格製品向けにスケールダウンすることができません。

100万個単位で製造した場合、Qt向けのSoCはWeb向けのSoCよりも1個当たりのコストが11ユーロ安く、しかも同様のUXを提供できる。

目次

1　はじめに

まずはHAM社の概要をご紹介しましょう。社名は機密性保持のために仮名にしていますが、世界有数の家電メーカーとしてよく知られた企業です。現在は年間数百万台のオーブンやガスレンジ、洗濯機、食洗機、冷蔵庫などを製造。製品の90%以上がOSを使わない、またはごくシンプルなリアルタイムOSを搭載したマイクロコントローラで制御されています。

製品のヒューマンマシンインターフェース（HMI）は物理的なノブ、ボタン、タッチパネル機能のないディスプレイ（多くは7桁表示、一部はTFTディスプレイ）で構成。高価格帯の製品にのみタッチディスプレイやOS（QNX、Linux）、SoC、マイクロプロセッサ（CPU）、グラフィカルプロセッサ（GPU）を搭載しています。SoCはシングルコアARM Cortex-A8を内蔵したもので、パフォーマンスとしては中の下に分類。GPUはOpenGL対応です。HAM社の高価格製品は製品全体の5%を占め、製造コストは他製品を2000ユーロ以上、上回ります。

HAM社は今後10年間で、高価格製品をマルチコアCortex-A9 SoCにシフトさせたいと考えています。ARMv8-A 64ビットアーキテクチャのSoCはコストがかかり過ぎるので現実的ではありません。同社はさらに中価格製品の大部分についても、シングルコアCortex-A5/A7/ A8またはARM11のSoCを搭載したいと考えています。低価格製品は、マイコン制御を継続する計画です。

またHAM社では、高品質な家電メーカーという評判も維持したいと考えています。製品ラインアップ全体でiPhone並みのUXの提供を目指しているのもそのためです。

2016年初めに、HAM社は新しいHMIとアプリケーションフレームワークの検討に着手しなければなりませんでした。使用中のフレームワークが製造中止になるからです。HAM社は直ちに、Web（ChromiumのレンダリングエンジンであるBlinkを使ったAngularJS HMI）とQt（Qt/C++を使ったQML HMI）の2つに候補を絞り込みました。

AngularJSはMITライセンスで、Blinkは主にBSDライセンスでそれぞれリリースされています。HAM社はWebテクノロジーならほとんどコストがかからないだろうと考えました。同じことはQtのLGPLv3ライセンスにも言えます。ところがHAM社の弁護士陣は、TiVo化対策である点を理由にLGPLv3ライセンスの使用に反対。つまりQtを導入するなら商用ライセンスが必須となり、開発者ごと、デバイスごとのライセンス料を支払わなければなりません。

Qtにとっては好ましくない状況の中、2017年初めにHAM社から連絡がありました。Webよりもかなり高コストになると思われるが、ひとまずQtの利点をプレゼンしてほしいというのです。世界中のあらゆる企業の購買部が納得するポイント、つまり数百万ドルのコスト削減を可能にする点について説明すれば必ず勝ち目があると思いました。そのためには、Webソリューションのコストの中でHAM社が見落としているもの、隠れた大きなコストを見つけなければなりません。そこで筆者は以下の仮説の証明に乗り出しました。

Webソリューションは、Qtソリューションよりもパワフルで高価なSoCがなければ、iPhone並みのUXを実現できない。

2　概説：ARM SoC

論点を分かりやすくするために、ここでARM SoCとは何かを概説しましょう。以下の表は、ARMコアを搭載したデバイスの例です。ARM9およびARM11はいずれもシングルコアで、32ビットアーキテクチャのARMv5およびARMv6をそれぞれ実行します。ARM9にはGPUがなく、一部のARM11にはGPUがあります。

Cortex-A8、A9、およびA15はいずれも、ARMv7-A 32ビットアーキテクチャを基盤とします。すべてGPUがあり、OpenGL対応です。Cortex-A8はシングルコア、A9とA15はマルチコアです。

ハイエンドのCortex-A57/53はARMv8-A 64ビットアーキテクチャを実行し、パフォーマンスは低価格帯のデスクトップPCクラスです。

表中の「大衆車のIVI（2017年）」のように一般名詞で記載したデバイスについては、機密保持契約の観点から具体的な製品名や社名を記載していません。

コア	デバイスの例
Cortex-A57/53	高級車の車載インフォテインメント（IVI、2015年）、Samsung GS6、Raspberry Pi 3
Cortex-A15	大衆車のIVI（2017年）、Samsung GS4
Cortex-A9	iPhone 4S、大衆車のIVI（2013年）、農耕機管理端末（2017年）
Cortex-A8	高級オーブン（2013年）、Nest製サーモスタット、iPhone 4、機内エンターテインメント（2014年）、農耕機管理端末（2013年）、Nokia N9
ARM11	Raspberry Pi 1、iPhone 3G、Nokia N8
ARM9	Nintendo DSi、Lego Mindstorm EV3、VoIP電話（2007年）

以下の表は、よく知られた産業グレードのSoCの1個、100個、1万個（10K）、100万個（1M）当たりの価格をまとめたものです（単位：ユーロ）。1個と100個の価格は、複数の電子部品販売会社のウェブサイトを参照しました。大口の価格は公表されていないので、1万個と10万個の価格については、個数が100倍になる毎に33%割引を適用した推定値を記しています。また100万個の推定値は、筆者自身が携わった複数のプロジェクトでの価格に概ね一致しています。

コンシューマー製品向けのSoCは、-30 °C～+70 °Cの極端な温度下では使用しない、高圧クリーナーや極端な埃への耐性が不要であるといった理由から、価格が著しく下がります。同じことは、コア数が少ないSoCにも当てはまります。たとえば産業グレードのi.MX6の場合、2コアの価格は4コアの約半分です。

コア	アーキテクチャ	コア数	1	100	10K	1M
R-CAR M3	Cortex-A57/53	4/4	202.50	135.00	90.00	60.00
TI AM5728	Cortex-A15	2	124.00	82.65	55.10	36.75
NXP i.MX6	Cortex-A9	4	71.35	47.55	31.70	21.15
NXP i.MX53	Cortex-A8	1	33.05	22.05	14.70	9.80
NXP i.MX35	ARM11	1	16.00	10.65	7.10	4.75
NXP i.MX25	ARM9	1	11.30	7.55	5.05	3.35

3　Web：スケールダウンが困難

Facebookは、iPhone 4S向けの自社アプリをWeb開発して満足度の高いUXを実現することができませんでした。iPhone 4SはデュアルコアCortex-A9 SoCを搭載しています。そこでFacebookは2012年に、スマートフォン用アプリの開発環境をWebからネイティブに移行し、以後はネイティブ環境を継続運用しています。HAM社は2017年にWebプロトタイプを構築し、同様の体験をしていました。

NetflixはWeb開発派です。同社は独自のレンダリングエンジンと高度に最適化されたReactJSを開発し、占有財産として運用しています。Netflixの場合は、Cortex-A9とまでは行かなかったもののCortex-A8を搭載したハイエンドデバイスで、iPhone並みのUX（30fps・110ms以内のレスポンス）をほぼ実現したことがあります。

NetflixはTVやSTB、BDプレーヤーで妥当なUXを提供するために数百万ドルを投じました。このレベルの成果を上げられるだけの優秀な開発者と予算を有している企業は、世界中を見渡してもごくわずかです。

3.1 Facebook – Web開発に投じた数百万ドルが無駄に

Disrupt SF 2012カンファレンスでのドルー・オラノフ（Drew Olanoff）氏とのインタビューの中で、FacebookのCEO／創設者であるマーク・ザッカーバーグ（Mark Zuckerberg）氏は数百万ドルの投資に失敗したことを認めました（Olanoff, 2012）。

「会社として過去最大の過ちは、ネイティブではなくHTML5に賭けたことだ。（中略）2年間を無駄にした」

ザッカーバーグ氏のコメントは、スマートフォン向けのFacebookアプリ開発に関するものです。この見解を基に同社はiOS／Android 向けアプリについて、Webからネイティブにシフトしました。ザッカーバーグ氏はこの方向転換の理由についてもコメントしています。

「モバイル（ユーザー）エクスペリエンスは、“十分優れている”程度では十分ではない。最高のクオリティを提供しなければならない。それを可能にする唯一の選択肢がネイティブへのシフトだ」

つまりFacebookが擁する世界有数のWeb開発者たちは、2012年にスマートフォンで優れたUXを提供できなかったのです。同社はiPhone 4sで、十分に優れたUXを実現できませんでした。iPhone 4sはクロック周波数800MHzのデュアルコアCortex-A9 SoC、マルチコアGPU、OpenGLグラフィックアクセラレーション、512MB RAMという仕様です（Phone 4S, 2011）。

W3C諮問委員会にFacebookの代表として参加しているトビー・ランゲル（Tobie Langel）氏は2012年に、技術的な課題について詳細を明らかにしています（Langel, 2012）。

- RAMの不足と、問題点を明らかにするために必要なツールの欠如：「最大の課題はメモリ関係だ。弊社のコンテンツのサイズを考えると、デバイスのハードウェア性能をアプリで使い切り、クラッシュが起きても不思議ではない。しかも弊社ではその原因を突き止められない」
- スクロール・パフォーマンス：「最も重要な課題の1つだ。ニュースフィードや、大量のコンテンツを延々とスクロールして見るタイムラインで特に問題が顕著だ」
- 「フレームレートが不安定で、UIスレッドに遅れ（スタッター）がある」
- OSの違いだけではなく、レンダリングエンジンの違いによっても問題が生じる：「ネイティブの慣性スクロールのフィーリングが、OSによって異なる。JSであるOS向けにカスタマイズすると、別のOSでおかしなフィーリングになる（不気味の谷が生じる）」

Facebookの失敗は5年以上前の話で、Web開発はその間に著しく改善したという反論もあるかもしれません。しかしHAM社の発見が、そうした反論が誤りであることを裏付けています。

2017年、HAM社はオーブンのHMIの重要な部分をWebとQtでそれぞれ開発しました。Web開発では、クアッドコアCortex-A9（NXP i.MX6）レベルのUXしか実現できませんでした。クアッドコアCortex-A9レベルのUXも許容範囲ではあるものの、優れているとは言えません。特に大きな問題点は、起動の遅さ、RAM消費量の多さ、アニメーション表示やスクロール時に起きるスタッターでした。

RAMについては、Webが512MBを消費するのに対し、Qtは64MBしか消費しないので、大きなコストの差が生じます。ボリュームが大きくなればその差は歴然でしょう。またWebではパフォーマンス上の問題点（メモリやスピードなど）を見つける優良なツールがないため、一時的に発生するエンジニアリングコストが著しく高くなるほか、デバッギングコストやプロファイリングコストがコーディングの何倍にも嵩むというリスクがあります。

3.2 Netflix – 思考錯誤のWeb開発

Netflixは2007年に主力事業をDVDレンタルからストリーミング配信にシフトした際、著しい細分化に直面しました。自社アプリをTVやSTB（セットトップボックス）、DVD／DVプレーヤー、ゲームコンソール、スマートフォン、タブレット、ラップトップ、デスクトップPCなどで運用する必要性が生じたのです。デバイス毎にCPUはローエンド（ARM9など）からハイエンド（Intel Core i7）まで差があり、GPUの有無の違いもあります。画面の解像度やフォーマットも大きく異なります。こうした事情はHAM社も同様ですが、Netflixの場合、ターゲットデバイスがよりパワフルであるという特色があります。

デバイス毎にアプリを開発するのは、Netflixには不可能でした。そこで同社はまず一部のデバイスに注力しましたが、事業を拡大するにはさらに多くのデバイスと顧客にリーチしなければなりません。対策として同社は2009～10年にかけ、Webへの移行を推し進めました。同社が選んだのはハイブリッドアプローチです。HMIはHTML5、CSS、-JavaScriptで書き、QtWebkitライブラリでレンダリングを行います。QtWebkitはブラウザと異なり、アプリからハードウェア性能へのダイレクトアクセスが可能です。ブラウザはサンドボックス環境であるため、ハードウェア性能へのアクセスが著しく制限されています。

2011年にNetflixの2人のエンジニアが、このアプローチが抱える問題点を解決する優れた方法を提示しました（McCarthy & Trott, 2011）。当時のNetflixアプリを知っている方なら、そのUXがどれだけひどい出来だったかも知っていることでしょう。FacebookのHMIと違い、NetflixのHMIはごくシンプルで、2～3個の横長のカバーと最大6個のイメージと同時に表示されるというものです。

このようなひどいUXでもNetflixが当時やっていけたのは、ライバルがいなかったおかげです。しかし2014年になると状況は一変。AppleやAmazon、HBOが新たに市場に参入してきました。NetflixのUXは競争力を失い、大幅に見直す必要性が生じました。

そこでNetflixは独自のレンダリングエンジンであるGibbonを開発。100%JavaScriptで書かれ、JITバージョンではないJavaScriptCoreで運用するエンジンです。
NetflixのエンジニアらはHTMI5のHMIを、ユーザーインターフェース構築用のJavaScript ライブラリであるReactJSですべて書き変えました（React, 2013）。また包括的なプロファイリングを行い、ReactJSを、Gibbonレンダリングエンジン用に高度に最適化した独自のReact-Gibbonへと改変しました。2014年当時にReactNativeがあれば、Netflixはそちらを採用していたでしょう。

多くのデバイスでは、110msのレスポンスタイムを実現するのも、30fps程度のさほど滑らかではないアニメーションを提供するのも極めて困難であるのは、ブログ記事 “Building the New Netflix Experience for TV”（Nel, 2013）や動画“Performance without Compromise”（McGuire, 2016）が指摘している通りです。一方でNetflixのエンジニアチームには、TVやSTB、DVD／DBプレーヤーに強力なCPUとGPUが搭載されており、フルHDに対応できるという利点もありました。たとえば第5世代のRokuプレーヤーはRaspberry Pi 3に匹敵する64ビットのクアッドコアARM Cortex-53という仕様です。

ここまでたどり着くのに、Netflixでは相当な作業工数と数百万ドル規模の多額の費用を投じたはずです。それでもなお、iPhoneが定めた100ms／60fpsという標準には到達できていません。

ここで、Netflixのように組み込みデバイスでWebアプリのパフォーマンスを最適化できる開発者が、世界中にごくわずかしかいない点も忘れてはいけません。普通のWeb開発者は、Netflixのアプリが実行される平均的なTVよりも400倍以上パワフルなコンピュータ向けにWebアプリを開発します。それでも大部分の開発者は、100msレスポンスと30fpsを実現するのに苦戦するのが実情です。

これを、HAM社のライバル社の体験と比較してみましょう。

> *ライバル社では、初心者と経験豊富な開発者という2人体制の開発チームでQtを用い、シングルコアCortex-A8を搭載したオーブン用のHMIを開発し、優れたUXを提供するのに、1年半を要しました。*

ちなみにこの開発チームは、独自のレンディングエンジンを書く必要も、独自のQMLを構築する必要もありませんでした。

Netflixのケースからは、（おそらく数百万ドル規模の）莫大な開発コストを投じなければ、Cortex-A9レベルのSoCで満足のできるUXを提供するのは困難であるということが読み取れます。Cortex-A9よりも性能が劣るSoCでは、まず不可能でしょう。

3.3 Web開発の組み込みアプリは稀

Webテクノロジーを使ってHMIに組み込みデバイスを開発している事例を、筆者はごくわずかしか知りません。その一例がNetflixです。ほかにはフランスの通信企業Orangeが開発したSTB「Livebox Play」があります（SoftAtHome, 2013）。一方で、TVやSTBのアプリの多くがWebテクノロジー（HTML5対応のHbbTVなど）を使って書かれています。

たとえば、ドイツ公営TVチャンネルのTVアプリがHTML5対応のHbbTVで書かれています。これらのアプリの問題点は、UXが劣っている点でしょう。

筆者の知る限り、Web開発を行っている家電メーカーは存在しません。エレクトロラックスはWeb開発でしたが、2011年にやめています。

自動車業界でも、Web開発を採用している例はわずかです。筆者が発見できたのも、ポルシェ918スパイダーに搭載されているインフォテイメントシステムの例のみで、これはHTML5を用いてHMIを開発しています（S1nn, 2014）。

AngularやReactNativeといったJavaScriptフレームワークは、組み込みデバイスではなく、モバイルやデスクトップ向けの開発に最適だと広告で謳っています。たとえばAngularは「モバイルとデスクトップ、ひとつのフレームワーク」が、ReactNativeは「一度学んだら、どこでも書ける。Reactでモバイルアプリを開発」がキャッチコピーです。スマートフォンのSoCが、典型的な組み込みSoCよりもかなりパワフルである点も忘れてはなりません。

4 Qt：スケールダウンが容易

一方、Qtで開発された多数の組み込みアプリは日々、数百万人の消費者に利用されています。このことから、Qtのスケールダウンがいかに容易であるかがうかがえると思います。エレクトロラックスはアプリ開発においてHAM社と同じ路線に進み、今やHAM社のライバルと言えますが、HAM社からはすでに5年の遅れを取っています。

4.1 Qt開発の組み込みアプリは多種多彩

以下の表は、Qtで開発された組み込みアプリの例を5つの業界（自動車、農業、航空電子工学、フィットネス、家電）別にまとめたものです。このほかにもQtは、工業オートメーションや医療技術の分野で広く応用されつつあります。他の多くの業界でのQt開発の成功事例は、公式ウェブサイトの「Built with Qt」ページでご確認いただけます（qt.io, 2017）。

				
▪OEMトップ15社のうち7社 ▪EVのOEM2社 ▪12社以上のティア1サプライヤー	▪Eジム ▪Eバイク ▪Precor	▪トップ3社のち1社 ▪ROPA ▪CCI ▪Krone ▪More...	▪OEMトップ3社のうち2社（パナソニックなど）	▪エレクトロラックス ▪HAM

これらのQt製品のSoCは、ARM11からCortex-A8、Cortex-A9、Cortex-57/53など多岐にわたります。

4.2 Qtの成功事例：エレクトロラックス

エレクトロラックスは2011～12年にHAM社と同じ路線に転換しました。同社はこの決定を、2014年にイタリアで開催されたQt Dayで発表しています（Penacchio, 2014）。エレクトロラックスはQtやWebを含め、複数のHMIおよびアプリケーション開発フレームワークを比較検討し、最終的にQtを選択しました。

「[2011年に]エレクトロラックスは、家電向けハイエンドユーザーインターフェースの開発におけるQt Quickの導入をはじめとし、グローバルにQtへの投資を行うことを決定した。（中略）Qtはエレクトロラックスのミッド／ハイエンド製品すべてについて、戦略的なUI開発プラットフォームとして活用できるだろう」

エレクトロラックスでは、Qtを開発に用いたオーブンをすでに1機種リリースし（Electrolux, 2015）、今後も家電製品にQtを活用していくとしています。オーブンの開発に2012～14年という期間をかけたことや、家電製品の利ざやの小ささを考えると、SoCはGPUを搭載したARM11かCortex-A8と思われます。動画を見れば分かるように、レシピ表示は非常にスムーズです。

またエレクトロラックスは、Qt活用のメリットだけでなく、デメリットも明らかにしました。

＋メリット	デメリット－
短時間で実装・習得できる	ライセンスの選択が難しい（商用vs LGPL）
他のHMIフレームワークよりもコストがかなり低い	PhotoshopやIllustratorなど、UIデザインツールとの統合が不可能（Qt 5.10では可能）
他のソリューションよりも堅牢かつパワフル	
GUIとバックエンドが明確に区別されている：デザイナーがGUIを変更できる	
C/C++との統合が簡単	
アニメーションが滑らか	
などなど	

エレクトロラックスはHAM社に遅れを取ること5年で、HAM社と同じ結論に達しました。

5　変化には常に時間がかかる

これまでWeb開発に否定的なさまざまな要因を見てきましたが、HAM社でWeb開発を支持する層からは次のような意見も出ています：「家電向けにブラウザベースのHMIがいずれ出てくるだろうが、まだしばらくは時間がかかるだろう。それと、SoCの仕様もおそらく異なるだろう」
この見通しが現実のものとなるには、Web開発は組み込みシステムに固有のいくつかの課題を克服する必要があります。

5.1 フラッシュメモリ、RAM、消費電力の問題

QMLレンダリングエンジンとJavaScriptエンジンを含む2つの共有ライブラリQt5QuickとQt5Qmlは、合計サイズが8MBになります（Ubuntu 16.04 LTS向け Qt 5.9.1の場合）。

WebレンダリングエンジンとJavaScriptエンジンを含む共有ライブラリQt5WebEngineCoreは、Qt 5.7.1で82MBです。5年前にリリースされた1バージョン前のライブラリQt5Webkitは15MBで、コンパイルスイッチを使って11MBまで圧縮できました。

これに対してChromiumブラウザの標準インストールは、Ubuntu 16.04 LTSの42MBを使います。QMLの5倍の容量が必要ということです。このようなWebのライブラリサイズの大きさは非常に重要なポイントとなります。

またWebアプリのRAM消費量は、Qtアプリを著しく上回ります。さらにWebアプリをQtアプリと同等のRAM消費量にしようとすれば、キャッシュミスが生じたり、フラッシュメモリからのページ読み込みが増えたりします。Webアプリのほうが、Qtアプリよりも遅いのです。

家電向けのQtアプリは、64MBフラッシュ／メインメモリのシステムで快適に動作します。一方、Webアプリは64MBでは足りず、おそらく256MBか512MBが必要です。大容量のフラッシュ／メインメモリが必要なWebソリューションのほうが、当然ながらコストがかかります。

HAM社のものも含め、多くのWebソリューションはWebエンジンとしてBlinkを使います。「デアリング・ファイアボール（Daring Fireball）」を運営するジョン・グルーバー（John Gruber）氏は同じMacBook Proと同じスクリプトを使い、ChromeとSafariでウェブページの読み込みをシミュレーション比較しました（Gruber, 2017）。バッテリーの持続時間はSafariが5時間30分、Chromeが3時間40分でした。これではWebソリューションは魅力的とは言えません。家電は省エネ性能が高くなければAAA格付けを得られないからです。

5.2 起動時間の問題

Chromiumブラウザは非常に重たいため、Webアプリの起動も遅くなります。クアッドコアIntel Core i7、2.5GHz、16GB RAM、高速SSDという仕様のハイエンドラップトップでも、Chromiumブラウザの起動には2秒以上かかります。Raspberry Pi 3では6秒です。Cortex-A8 SoCでChromiumを10秒以内に起動させるには、長時間の最適化作業が必要であり、しかも確実に成果が出せるとは限りません。また上記の起動時間には、OSの起動時間やWebアプリのローディング時間は含んでいません。

以上をQtの場合と比べてみましょう。クアッドコアARM Cortex-A9（1GHz、1GB RAM、8GBフラッシュ）でLinuxおよびQMLインストルメントクラスタを起動させると、いずれも1.5秒かかります（Avila, 2016）。アプリの起動にはQMLで0.25秒、Linuxで1.25秒かかります。ハイエンドラップトップでChromiumブラウザを起動させるよりずっと速いわけです。
オーブンやガスレンジ、プリンタ、STB、TVのユーザーは、電源を入れて即座に立ち上がるのが当然だと考えます。起動に3秒以上かかれば、購入を検討中の消費者はすぐさまその製品を候補から外すでしょう。

QMLアプリの起動の速さに著しく貢献しているのが、Qtライブラリ間の静的リンクとQMLコンパイラです。これらの特性は、Qt商用ライセンスでしか利用できません。

静的リンクはライブラリからアプリに不要な部分を削除するので、実行ファイルのサイズをさらに小さくできます。OSの必要なアプリを実行する場合、フラッシュメモリから実行ファイルと関連するすべてのライブラリをロードしなければなりません。フラッシュメモリからの読み込みには、RAMからの読み込みより非常に多くの時間がかかります。したがって、実行ファイルとライブラリが大きければ大きいほど、起動時間も長くなります。

ただし、静的リンクを行ってもChromiumブラウザは42MBで、共有QMLライブラリの8MBには及びません。さらに、静的リンクを行ってもWebの実行ファイルはQtの実行ファイルの5倍の大きさがあります。つまりWebアプリのローディングには、Qtアプリの5倍の時間がかかる計算になります。

QMLコンパイラはQMLコードをC++コードに翻訳し、翻訳後のコードがC++コンパイラによって機械コードに翻訳されます。QMLコンパイラはまた、JavaScriptの関数や式の変換も行います。つまり、JavaScriptエンジンのジャストインタイム(JIT)コンパイルが、ランタイム中ではなく、コンパイルタイム中に行われるという仕組みです。

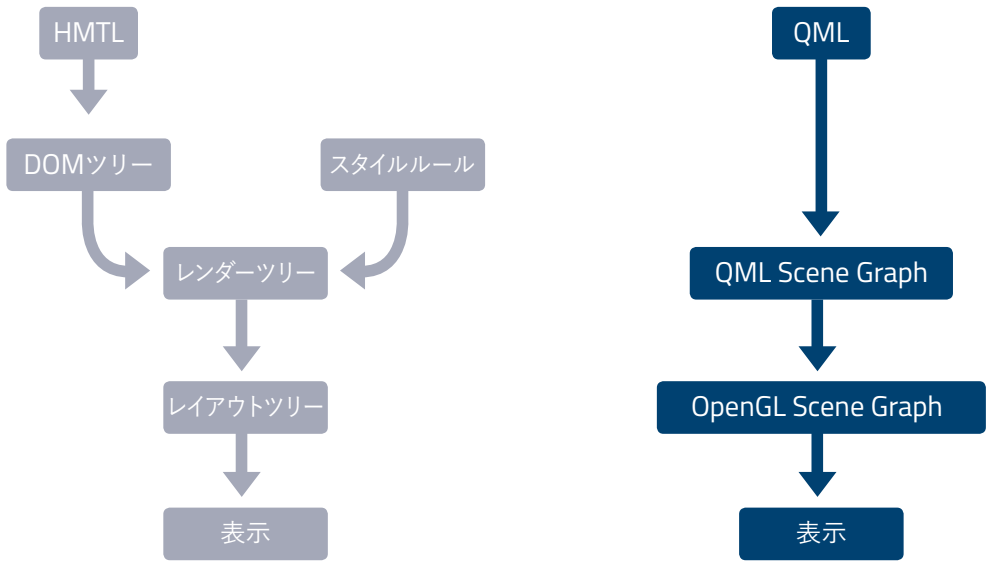
QMLコンパイラを使うだけで、収穫機管理端末のQMLアプリの起動時間を30%短縮化するのに成功しました(Stubert, 2016)。

Webには、HTML/CSS/JavaScriptコンパイラのようなものはありません。HTML、CSS、JavaScriptの難解な標準に100%即したコンパイラを書くのが非常に困難なためです。仮に書けたとしても、WebがQMLに追いつくには数年間かかるでしょうし、その間もQMLは進化し続けているはずです。

また、もし仮にQMLコンパイラがなかったとしても、WebはQMLに太刀打ちできないでしょう。JavaScriptエンジンはここ数年間で著しくパフォーマンスが改善しているからです。Qtを採用する開発者の場合、同じテクニックを使ってJavaScriptエンジンの高速化を図ることができます。またQMLは、最適化が容易であるというメリットも備えています。QMLはHTMLやCSSよりもシンプルな言語で、Qtの開発者チームはQMLを100%管理しているからです。

5.3 レンダリングフローの問題

下図はWeb(左)とQML(右)のレンダリングフローを比較したものです。



Webのレンダリングフローの概要を以下にご紹介しましょう（詳細はGarsiel & Irish, 2011を参照のこと）。

- ステップ1：HTMLパーサーがHTMLドキュメントからDOMツリーを作成。HTMLの文法はコンテキストフリーではないため、パースが困難。
- ステップ2：CSSパーサーがスタイルシートからスタイルルールを作成。
- ステップ3：スタイルルールがDOMツリーのノードに適用される。カスケードスタイルルールなので、どのルールがどのDOMノードに適用されるのか見極めるのが困難。そのためこのステップでは、レンダリング順に並んだビジュアルノードのレンダーツリーが作成される。
- ステップ4：レイアウトステップで各ノードのポジションとサイズを算出。
- ステップ5：ペアリングステップでノード毎にレンダーツリーノードをトラバースし、表示される各ノードをペイント。

ステップ1、2、3が最もコストのかかるステップです。中でも、カスケードスタイルルールを適用するステップ3にコストがかかります。CSSルールを1つ適用する度に、レンダーツリーにコスト高な変換を実行しなければなりません。ステップ4と5は、WebとQMLで共通です。

Netflixは最初の3つのステップについて、根本から最適化を図っています。HTML5とCSSをできる限り使用しないことで、CSSルールの数とレンダーツリーのサイズを削減しています。また複数のアルゴリズムを適用して、サイズの一層の縮小化も図っています。これと同様のアプローチを取っているのがReactNativeです。

したがってQMLフローでは、Webの最初の3つのステップをわずか1つのステップで完了します。

QMLはコンテンツ（HTML）とスタイル（CSS）を分けていません。したがってQMLフローでは、Webの最初の3つのステップをわずか1つのステップで完了します。またQMLフローでは、CSSスタイルルールをDOMツリーに適用する、コスト高なステップ3が不要です。さらに、Open GL Scene Graphへのダイレクトかつシンプルなマッピングを実行できるよう、高度に最適化されています。

企業がWebソリューションを選べば、社内の開発チームは常に最適化を図りながら、極めて特別な方法でアプリケーションコードを書かなければなりません。さらに、レンダーツリーを最適化するためにWebレンダリングエンジンを変更する必要性も生じます。

そうしてWeb開発チームがコードやツールの最適化を図っている間に、QML開発チームはアプリケーションコードを完成させることが可能です。Sequality社の実験も、この事実を裏付けています（Larndorfer, 2017）。同社の開発チームは、WebとQtで同じ開発時間を投じ、後者でより大きな成果を上げました。またQtソリューションのほうがWebソリューションよりもスムーズでレスポンス性に優れていたと指摘しています。

6 結論

FacebookとNetflixのストーリーからは、クアッドコアCortex-A9 SoCでWebアプリを実行してもiPhone並みのUXを提供できないことが分かります。またNetflixの事例からは、この開発プロジェクトに数百万ドルのコストが必要であることが読み取れます。

HAM社は上記の信ぴょう性を確認するべく、既存のHMIの重要な部分についてWebとQtの両方でプロトタイピングを行いました。その結果、クアッドコアCortex-A9 SoCでは満足のいくUXを実現できないことが判明しました。しかもこのレベルのパワフルなSoCを使用しても、Webソリューションは起動時間が長く、莫大なメモリを消費し、スクロールやアニメーションがスタッターすることもあるのです。HAM社がWebソリューションを導入すれば、SoCコストが嵩むだけではなく、ソリューションの最適化に相当な時間と費用と投じなければならないでしょう。

Qtで開発された製品、たとえばエレクトロラックスのオーブンやインフォテインメントシステム、収穫機管理端末、機内エンターテインメントシステムはいずれも、シングルコアのCortex-A8 SoCでiPhone並みのUXを実現しています。Qtにはスケールダウンが容易であるというメリットも備わっています。Raspberry Pi 1のようにGPU搭載のARM11 SoCでも同様に、iPhoneレベルのUXを提供可能です。GPUを搭載しないARM11 SoCやARM9 SoCでも、安いSoCで十分に優れたUXを実現する、という目標の妥協点を見い出すことができます。

Webソリューションは、クアッドコアCortex-A9 SoC以上でなければiPhone並みのUXを実現できない。これに対しQtソリューションは、シングルコアCortex-A8 SoCで上記を実現できる。

第2章のSoCの価格をまとめた表を基に、WebとQtそれぞれの、10万個（10K）当たり、100万個（1M）当たりの合計コストを算出することができます。

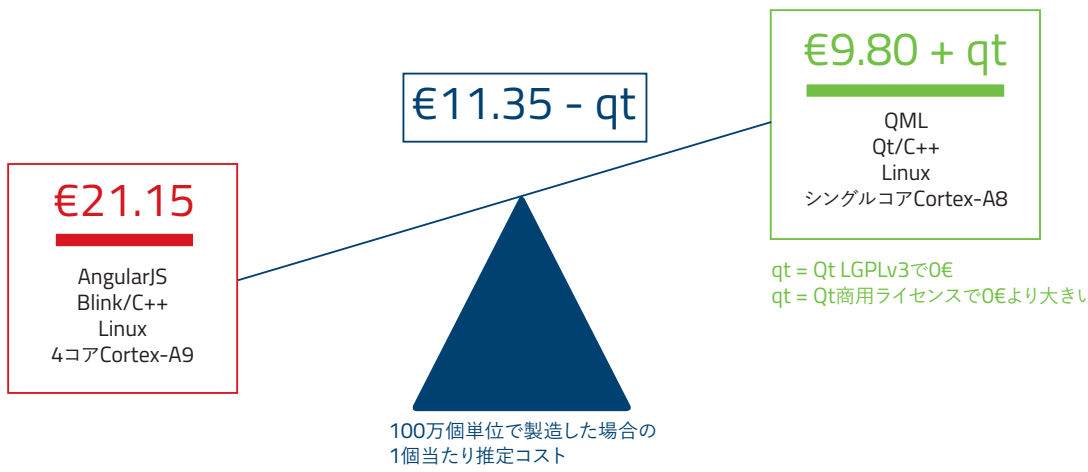
	€ / 10k当たり	€ / 1M当たり
Web: Cortex-A9 NXP i.MXP6 quad	3 170 000	21 150 000
Qt: Cortex-A8 NXP i.MX53	1 470 000	9 800 000
コストの差 (€)	1 700 000	11 350 000

個数別のSoCのコストは、Qtのほうがそれぞれ€1,700,000と €11,350,000低い計算です。つまり、Qtはハードウェアコストを約53%削減できるのです！

下図は100万個単位で製造を行った場合の、1個当たりコストの差を示したものです。変数「qt」は、商用ライセンスの1個当たりコストを表します。Qt LGPLv3で、qt=0です。Qt商用ライセンスでは、qtはゼロより大きくなります。

同じアプリケーション

（例：家電、プリンタ、STB、TV、機内エンタメシステム、機器類管理端末）



変数qtは11ユーロのコスト差のごく一部を占めるだけなので、HAM社はQtを導入することにより数百万ユーロのコストを削減できます。こうしてHAM社は、Qtを選択するという決断を容易に下しました。

7 Sources

Avila, R. (2016, 04 20). Fast-Booting Qt Devices, Part 1: Automotive Instrument Cluster. Retrieved from qt.io: http://blog.qt.io/blog/2016/04/20/fast-booting-qt-devices-part-1-automotive-instrument-cluster/

Electrolux. (2015). Ovens Electrolux Color Touch Screen. Retrieved from Youtube: https://www.youtube.com/watch?v=BpI7dku7Yr8

Garsiel, T., & Irish, P. (2011, 08 05). How Browsers Work: Behind the scenes of modern web browsers. Retrieved from Html5 Rocks: https://www.html5rocks.com/en/tutorials/internals/howbrowserswork/#Rendering_engines

Gruber, J. (2017, 05 24). Safari vs. Chrome on the Mac. Retrieved from Daring Fireball: https://daringfireball.net/2017/05/safari_vs_chrome_on_the_mac

iPhone 4S. (2011). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/IPhone_4S

KDAB. (2017, 09 28). CCI – putting intelligence into agriculture, using Qt. Retrieved from KDAB: https://www.kdab.com/cci-putting-intelligence-agriculture-using-qt/?utm_source=Master+List+06-16&utm_campaign=d-ca3188043-EMAIL_CAMPAIGN_2017_09_25&utm_medium=email&utm_term=0_bdde4cdc11-dca3188043-101570617

Langel, T. (2012, 09 15). Perf Feedback - What's slowing down Mobile Facebook. Retrieved from W3C: http://lists.w3.org/Archives/Public/public-coremob/2012Sep/0021.html

McCarthy, M., & Trott, K. (2011, 07 29). Netflix Webkit-Based UI for TV Devices. Retrieved from Slideshare: https://www.slideshare.net/mattmccarthy_nflx/netflix-webkitbased-ui-for-tv-devices-9168822

McGuire, S. (2016, 03 23). Performance without Compromise. Retrieved from Netflix Technology Blog: https://medium.com/netflix-techblog/performance-without-compromise-40d6003c6037

Nel, J. (2013, 11 18). Building the New Netflix Experience for TV. Retrieved from Netflix Technology Blog: https://medium.com/netflix-techblog/building-the-new-netflix-experience-for-tv-920d71d875de

Olanoff, D. (2012, 09 11). Mark Zuckerberg: Our Biggest Mistake Was Betting Too Much On HTML5. Retrieved from Techcrunch: https://techcrunch.com/2012/09/11/mark-zuckerberg-our-biggest-mistake-with-mobile-was-betting-too-much-on-html5/

Penacchio, N. (2014). Next-gen appliance e Qt Quick in Electrolux. Retrieved from Youtube: https://www.youtube.com/watch?v=bnRYKNPf2x0

qt.io. (2017). Built with Qt. Retrieved from qt.io: https://www1.qt.io/built-with-qt/

React. (2013). Retrieved from ReactJS: https://reactjs.org

S1nn. (2014, 05 06). Porsche 918 Spyder Infotainment System Is Based on HTML5 Technology. Retrieved from Pddnet: https://www.pddnet.com/news/2014/06/porsche-918-spyder-infotainment-system-based-html5-technology

SoftAtHome. (2013). Orange Livebox Play. Retrieved from SoftAtHome: https://www.softathome.com/pages/customer-success-stories

Stubert, B. (2016, 11 20). 30% Faster Startup Thanks to QtQuick Compiler. Retrieved from Embedded Use: http://www.embeddeduse.com/2016/11/20/30-faster-startup-thanks-to-qtquick-compiler/