sequality
SOFTWARE
ENGINEERING

HTML5 vs Qt
フルソフトウェアスタックの比較

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

概要

HTML5とQtはいずれもずば抜けたテクノロジーであり、新たな開発プロジェクトにどちらを使うか決めるのは容易ではありません。フルスタックの開発フレームワークであるQtと、ブラウザ用HTMLアプリケーション実装の標準であるHTML5を比較するのは、リンゴとオレンジを比べるようなものだからです。

本ホワイトペーパーでは、HTML5アプリケーションおよびそのバックエンドソフトウェアレイヤーと、フルスタックQtアプリケーションのアーキテクチャを比較します。具体的には、ソフトウェアアーキテクチャの違いと、それらの違いがアプリケーションと総合的な製品戦略にどのような影響を及ぼすかを明らかにします。また、開発エコシステムにも広く目を向け、サードパーティが提供しているツール、開発コミュニティによるコントリビューション、技術的な選択肢がターゲットハードウェアの選択に及ぼす影響といったポイントについても解説します。

本ホワイトペーパーの目的は、どちらがより優れたテクノロジーかを決めることではありません。どちらのテクノロジーが既定の環境に最も適しているか、両者を組み合わせるのが望ましい環境とはどのようなものか、オープンかつスケーラブルなアーキテクチャの開発プランをいかにして策定するべきか、といった疑問を解くための知識を提供することが目的です。また本ホワイトペーパーは、組み込みデバイスと産業用ディスプレイパネルに焦点を当てています。基本的な情報はデスクトップやモバイル環境にも応用できますが、一部の詳細な情報については組み込みデバイスや産業機器により適した内容となっています。

> 「万能薬は存在しません。環境によって、明らかにQtが適している場合もあれば、HTML5が適している場合もあります。ふさわしい技術を選択するに当たっては、アーキテクチャ全体、つまりフルスタックをベースに検討しなければなりません」

目次

HTML5 vs Qt –不完全な比較

Webアプリか、それともネイティブアプリか。どちらが最良のHMI開発につながるかという議論において、しばしば置き去りにされるのがバックエンドの問題です。フロントエンドテクノロジー（HTML5やQt Quick／QML）は当然ながら制御ソフトウェアの重要な部分を担っており、これらがなければ視覚化はできません。とはいえ、データを制御系に送る、CAN BUSなどのシリアルインターフェースを統合する、データベースにデータを保存するといった役割を果たすバックエンドテクノロジーがなければ、せっかくのHMIもただの飾り物になってしまいます。

Qtは「フルスタックオファー」であり、一方HTML5はブラウザ用HTMLアプリケーション実装の標準です。そのため、HTML5はブラウザやWebサーバ、エンジンなどのコンポーネントと組み合わせなければ、ほかの機能を実現できません。従って本ホワイトペーパーでは、これらのコンポーネントについても比較を行っています。

アーキテクチャの比較：機能的要件

ソフトウェアアーキテクチャのデザインに当たってはまず、そのソフトウェアにどのような機能を持たせるかを事前に決定する必要があります。話をシンプルにするために、ここでは産業、医療および自動車といった分野を例にセンサーデータを表示し、マシンの行動および状態に関する情報をユーザーに提供する、以下の2つの基本的機能を例に考えてみましょう。

- タッチスクリーン型の組み込みLinuxデバイスでGUIを表示する
- UIのインプットを処理し、ハードウェアインターフェース（CAN BUS、Ethernet、GPIOなど）とデータをやり取りする

デバイスとインタラクションする主な方法は2つあります。内蔵のタッチディスプレイで直接、もしくはスマートフォンやタブレット、PC経由でリモートにインタラクションする方法です。重要機能はすべてインターネット接続なしでも利用できなくてはならないので、デバイスがローカルにデータの処理や視覚化を行ったり、ハードウェアと直に接続できることが大切です。例としては、産業機器に内蔵された制御用ディスプレイ、車載インフォテインメントシステム、患者を監視する医療用デバイスなどがあります。

HTMLとQtの典型的なアーキテクチャの比較

図1および2は、バックエンドとフロントエンドを含めた最も基本的なLinuxベースのアプリケーションアーキテクチャをHTML5とQtのそれぞれを示したものです。

HTML5アーキテクチャは、2つの必須要素で構成されています。Webサーバ+エンジンと、Webブラウザで実行されるHTML5アプリケーションです。HTML5アプリケーションはプレゼンテーションレイヤーを担当し、Webサーバ+エンジンはビジネスロジックの実行やローカルデータの処理、Linux OSの標準アクセス層経由でのハードウェアインターフェース（CAN BUSなど）との接続を担います。

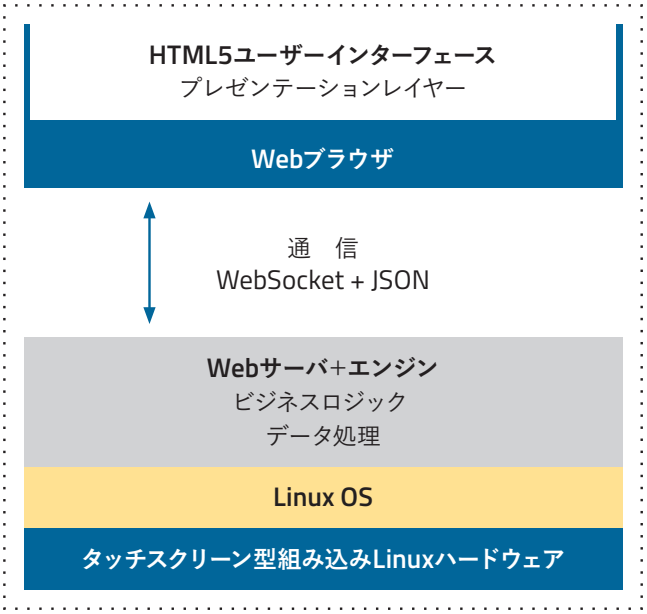

図1：HTML5ウェブアプリケーションの基本的なアーキテクチャ

HTML5ソフトウェアアーキテクチャ

アーキテクチャ図は、使用するテクノロジー別に色分けしています。

HTML5ユーザーインターフェース：プレゼンテーションレイヤーはWebテクノロジーを使用します。図1は簡潔に「HTML5+JavaScript」を想定していますが、実際のプレゼンテーションレイヤーはより多様なテクノロジーを用いることになります。近代的な実装では、AngularやReactのようなシングルページアプリケーションフレームワークのアプローチを取ることが多く、パッケージ経由で機能を追加できます。

Webブラウザ：デバイスのブラウザはWebアプリケーションを実行します。今日最も一般的なブラウザはC++で書かれていますが、組み込みLinuxシステム向けのオープンソースパッケージも利用可能です（YoctoやBuildroot）。通常、既存の実証済みブラウザのソースコードに変更は加えないものです。変更するとなると、ブラウザの複雑なソースコードパッケージの保守も行わなければならず、開発リソースが大幅に膨れてしまうためです。

Webサーバ+エンジン：WebサーバはWebアプリケーションをHTMLファイルとしてブラウザに送ります。ここでは、データをWebアプリケーションに提供／送信する環境も必要です。方法としては、Webサーバ環境がこれらを行う（Node.jsのモジュールを使い、JavaScript経由でCAN BUSデータアクセスを実行）、もしくはC++、Java、その他の言語で別途サーバプロセス（＝エンジン）を経由し、Web互換の通信プロトコル（WebSocket+JSONなど）でデータを送信するという選択肢があります。

このセクションは実際にはもっと複雑になることもあります。ハードウェアインターフェースや特殊な通信プロトコル（OPC-UAなど）を用いた場合、Webサーバ以外のテクノロジーも必要になるからです。

Linux OS：ここでのシナリオはいずれも、組み込みLinux OSの使用を想定しています。この層はビルドシステム経由で生成されるもので、ドライバを含み、上の層を実行するソフトウェアモジュールも必要です。

ハードウェア：OSはハードウェア上で実行します。通常、ハードウェアは近代的なCPU（i.MX 6デュアルコアなど）、CAN BUSのコネクション、Ethernet／WLAN、タッチディスプレイで構成されます。

まとめると、組み込み製品でWebアプリケーションを開発する場合、開発者は少なくとも4つのテクノロジー（HTML5 UI、ブラウザ、Webサーバ、エンジン）を使用する必要があります。

Qtソフトウェアアーキテクチャ

図2のように、組み込みデバイスの最も基本的なQtアーキテクチャは単一のブロックで構成されます。

図2：Qt Quickアプリケーションの基本的なアーキテクチャ

Qt Quickオールインワンアプリケーション：より複雑なアプリケーションでこのアーキテクチャを使うことはないはずです。とはいえ適切なソフトウェアアーキテクチャを選べば、このアプローチでも組み込みデバイスに大きなメリットをもたらすことができます。1つのアプリケーションプロセスで多くのアスペクトやレイヤーをカバーできれば、その分、アプリケーションメモリのオーバーヘッドや消費量を減らすことができます。たいていの場合、このレイヤーの内部ソフトウェアアーキテクチャには、プレゼンテーションレイヤー（Qt QuickやQMLで実行される）とC++のビジネスロジックレイヤーが含まれます。

Qtベースのアプリケーションを使うシステムでも、高い拡張性と柔軟性を確保するためにHTML5ベースのシステムと同様のアーキテクチャを採用することがよくあります（図3参照）。プレゼンテーションレイヤーは別のアプリケーション（Qt Quick／QML）にアウトソースし、別のプロセスで実行されるビジネスロジックレイヤーとWebSocketで通信する仕組みです。このようにプロセスレベルでプレゼンテーションレイヤーとビジネスロジックレイヤーを分けるアプローチには、多くのメリットがあります。

- 必ずしもエンジンと共通のハードウェア上でGUIを実行する必要がない。
- 1つのエンジンにGUIインスタンスをいくつでも接続できる。
- 別のアプリケーション（QtWebAssemblyなど）でシステムを遠隔制御するのが簡単。

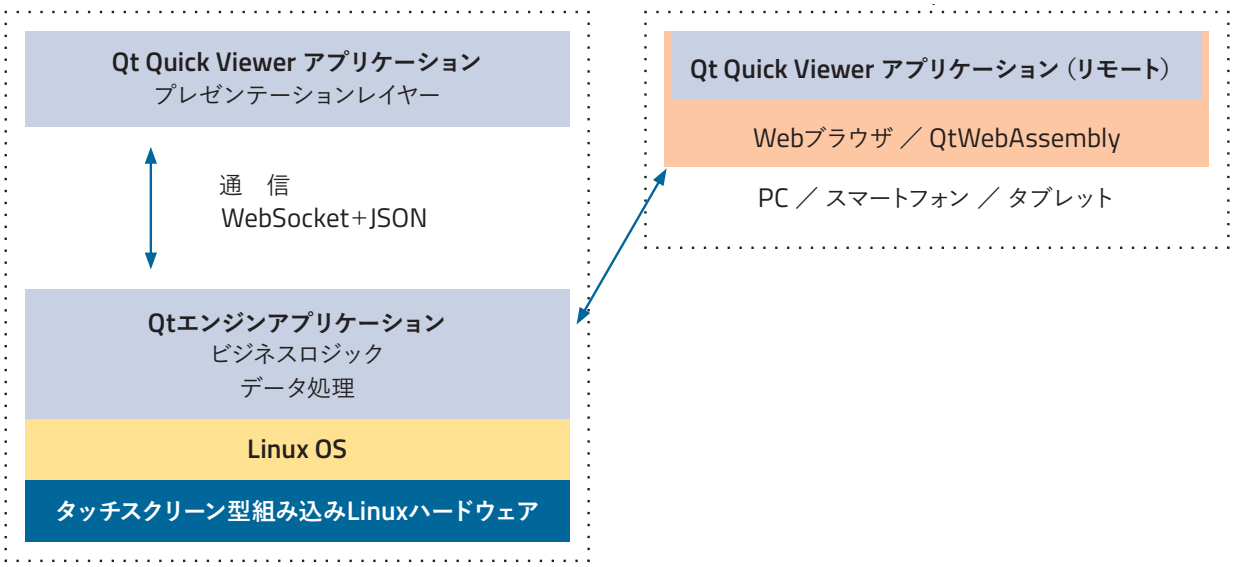

図3：スケーラブルなQt Quickアプリケーションのアーキテクチャ

Qt Quick Viewerアプリケーション：このレイヤーにはプレゼンテーションレイヤー（Qt Quick／QML）といくつかのC++クラスがあります。アプリケーション構造の完全性を確保し、ビジネスロジックレイヤーと通信（WebSocket+JSONなどを経由）するレイヤーです。

Qt エンジンアプリケーション：ビジネスロジックレイヤーとデータアクセスレイヤーで構成され、ハードウェアへのアクセスをすべて担います。多くの場合、このレイヤーはQtのC++クラスを実装し、Qtフレームワークが提供する多数の優れたハードウェアインターフェースを活用します。図3のQtエンジンアプリケーションは、図1のWebサーバ+エンジンに相当します。

Qt Quick Viewerアプリケーション（リモート）：Qt 5.12以降では、QtWebAssembly経由でQt Quickアプリケーションを配布し、Webブラウザを仮想マシンとして実行できます。マシンをローカルでも（タッチスクリーンなどを使う）リモートでも制御したい場合に便利なアプローチです。Qt Quick Viewerアプリケーションと Qtエンジンアプリケーションを分けることにより、リモートのViewerアプリケーションに同じコードを使うことができます。

リモートQt Quick Viewerアプリケーションの下のWebブラウザは、組み込みデバイスのソフトウェアスタックに属しません。また保守についても、通常はシステムメンテナンスの一環としてPCやスマートフォン、タブレットのユーザーが行います。従って、ソフトウェアアーキテクチャで使用するテクノロジーはLinux OSのほかにはQtだけとなり、保守作業や依存性の特定が簡素化されます。

また、プレゼンテーションレイヤーやビジネスロジック、データ処理などすべてに同じフレームワークを使える点も大きな強みです。さらにQtフレームワークはCAN BUS、Modbus、Ethernet、Bluetoothといった標準的なハードウェアインターフェースに対応するほか、MQTT、OPC-UA、XMLパーサー、JSONパーサーなど多数の便利な通信規格とも互換性があります。そのためQtフレームワークでは同種のアーキテクチャを活用でき、他のフレームワークへの依存を回避することができます。

リンゴとリンゴを比べる：フルスタックエコシステムでの比較

言うまでもなく、フルスタックでの比較を行わなければ意味のある結果は得られません。フルスタック、つまり「Qt」と「HTML5+Webブラウザ+Webサーバ+エンジン」の比較です。続く各セクションでは、システムはどうあるべきか、その方向性を左右する決定的要因について説明していきます。

HTML5：ブラウザという名の安全な「サンドボックス」から抜け出すには

ブラウザはUIの一部であり、安全な環境（サンドボックス）でHTML5アプリケーションを実行します。ただし、この安全なサンドボックスから抜け出して以下のような機能を実装する方法はないのだろうか、という疑問が生まれることも少なくありません。

(1) 触覚ハードウェアボタンやロータリープッシュコントロールの統合

(2) 非アクティブディスプレイの輝度を下げるなどの省エネ機能

(3) USB経由または無線でのシステム全体のソフトウェアアップデート

(4) ハードウェアアクセラレーテッド動画デコーディング＆再生

(5) OpenGLと内蔵グラフィックカードによる滑らかなキネティックスクロール

1〜3については、Webサーバのエンジンに実装する（+HTML5にスピーディに通知する）という方法で実現できそうですが、4と5についてはブラウザ環境に頼るほかありません。

サンドボックスから確実に抜け出す方法としては、アプリケーション専用のブラウザをディスプレイパネルに用いることが可能です。これによりHTML5ブラウザにリンクしたハードウェアに直接アクセスすることができます。具体的なやり方としては、Qtコンポーネントの1つであるQtWebViewを使うアプローチがあります。これは、Qtフレームワークに含まれる100%カスタマイズ可能なChromiumエンジン基盤のWebブラウザコンポーネントで、組み込みデバイスでも、大型の産業用ディスプレイパネルでも使えます（「HTML5とQtWebView」のセクションも参照）。このアプローチなら、ブラウザの大規模かつ複雑なソースコードとのダイレクトインターフェースの代わりにQtWebViewの高度なインターフェースを用いて、サンドボックスから抜け出すことが可能です。

HTML5：成長し続ける開発者ツール

Webフレームワークの人気の高まりと共に、IDE、ブラウザベースのコードジェネレータ、デバッギングツールなど、その開発ツールに対する注目度も高まりを見せています。これらの開発ツールは、HTML　およびJavaScriptコードのシンタックスハイライトやシンタックスエラーチェックといった一般的な機能を提供するだけではなく、コード分析やコード補完、一部のWebフレームワークでのボイラープレート生成といった機能、Angular、vue.js、Node.jsといったテクノロジーなども備えています。

FirefoxやChromeをはじめとする近代的なブラウザは、内蔵のデバッギングツールを使って必要に応じローカルでデバッギングを行うことができます。あるいは、VS CodeやWebStormといった初歩的な開発環境をブラウザと一緒に使うこともできます。

まとめると、静的型付け言語（Visual Studio、Eclipse、IntelliJ IDEAなど）であれば、Web開発ツールはメジャーなIDEと同様の威力を発揮するようになってきたと言えます。

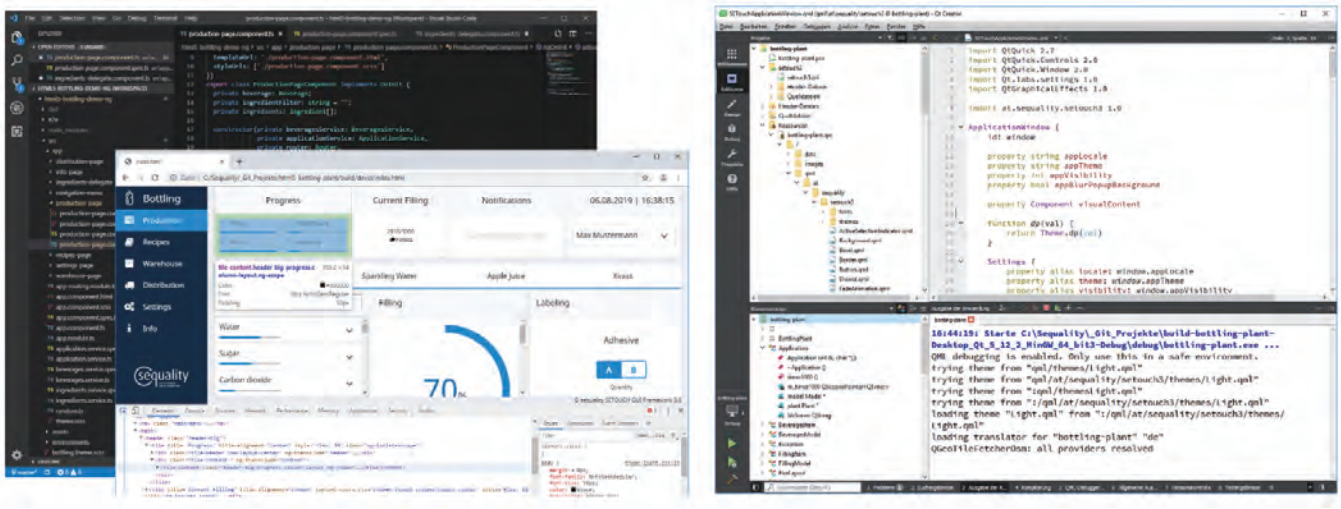

図4：左）バックグラウンドにVisual Studio Code を、Google ChromeにDOM Inspector をそれぞれ用いたHTML5開発環境。
右）右上端にQMLコードを、下にアクティブなアウトプットウィンドウを置いたQt開発環境。

HTML5：製品ライフサイクルを通した保守

スマートフォンやPCのWebアプリケーションは、ソフトウェアサービスプロバイダにとって大きな強みとなります。アプリケーションのランタイム環境、つまりブラウザの保守の心配がいらないからです。保守作業には定期的なブラウザ／OSアップデートが含まれますが、これはエンドユーザーに「アウトソース」されます。閉環境で使われているブラウザであれば、追加的なテストはさほど手間はかかりません。スマートフォンやタブレット、デスクトップパソコンのように開環境であれば、テストはソフトウェアライフサイクルを通した保守に欠かせない作業となります。

ブラウザのほかにも、HTML5フレームワークとJavaScriptライブラリのランタイム環境への追加や保守業務が必要です。これらのコンポーネントは、ライブラリの背後にあるGoogle（Angularなど）、Facebook（Reactなど）、Microsoft（TypeScriptなど）のようなWeb大手のプラットフォームの影響で、頻繁に変更やアップデートが行われます。こうしたWeb大手のイノベーションサイクルは、一般的なマシンやデバイスの製品ライフサイクルより早いことが少なくありません。また、ソフトウェアWebライブラリによっては長期サポートがない場合があります。つまり、JavaScriptフレームワークが向こう10年間にわたってサポートされるという保証はありません。

どのWeb環境を選ぶにしても、強力なHTML5およびJavaScriptライブラリへの変更に備えておくか、今後5年間で新しいものに交換しておくべきです。最近のAngularJSやAngularの例からも明らかなように、市場に出回ってすでに数年が経つようなWebフレームワークでも、数カ月後には互換性のない変更が加えられたり、古いバージョンが廃れたりすることはあります。

Qt：製品ライフサイクルを通した保守

Qtフレームワークも世界中に広がる大規模コミュニティによって常に保守、改善、開発が行われています。従って、製品のソフトウェア保守戦略は不可欠です。ただしWebエコシステムと異なり、Qtフレームワークの多くの部分（特にC++クラス）はQt 4（2005年リリース）から変動がありません。またWebテクノロジーと違い、Qtは長期サポート（LTS）バージョンを提供しています。定期アップデートではセキュリティホールや著しいエラーの修正を行い、アプリケーションコードの変更や交換は不要です。ソフトウェアのマイナーアップデート（Qt 5.10 → Qt 5.11など）でも、ほとんどのケースでソースコードの変更は不要です。メジャーアップデート（Qt 4 から Qt 5）もこれまで、ソースコードの調整はなし、あるいはシンプルなテキスト交換で済んでいます。Qt 6へのアップデート計画についても、ソースコードの互換性を維持して実施できる見通しです。

HTML5：クロスブラウザテストとブラウザ間の微妙な差異

HTML標準は過去10年間で改善してきましたが、Webアプリケーションは依然として各種ブラウザでのテストが必要です。3つの主流ブラウザ（Google Chrome、Mozilla Firefox、Apple Safari）には小さな差異があり、Webアプリケーションが異なる挙動をする要因となっています。これらの差異は組み込みデバイスのアプリケーションには目立った影響を及ぼしませんが（単一のローカルブラウザを使うため）、スマートフォンやタブレット、PC経由でデバイスを遠隔操作すると問題が発生します。

また、使用するブラウザが1種類だけの場合でも、Webアプリケーションはバージョンアップデート後に正しく挙動する保証がありません。Webアーキテクチャでは、複数のブラウザと複数のバージョンでテストするのが常に必須と言えます。

極細部のコントロール

極細部のコントロールは必ずしも必要ではありません。宣言型スクリプト言語（QML、JavaScript、HTML5など）がGUIを書く際のデフォルトになっているのもそのためです。とはいえ、特に組み込みシステムでは、極細部までアクセスしてシステムを100%コントロールする必要が生じることもあります。Qtならこれが可能です。Qt Quickアプリケーションは組み込みシステムを100%コントロールできるからです。Webアーキテクチャのシステムの場合、視覚化に関して言えば100%のコントロールは困難です。3Dアクセラレーションや特殊なハードウェアアクセラレートされたエフェクトが必要な場合や、グラフィカルな遷移を最適化したい場合なども、ブラウザの定義済みのデバッギングオプションに縛られることになります。もしくは、複雑なC++ブラウザの深部にアクセスするしかありません。

C ++ は若手開発者には難しい?

要件に最適なテクノロジーが何であれ、最終的にはそのテクノロジーで仕事ができる開発者が必要であり、現在チームに所属している従業員の能力開発やトレーニングが欠かせないものとなります。

これはHTML5とQtの両方に言えることですが、若手の開発者や意思決定者はC++を避ける傾向があるようです。理由はいろいろあります。ベテラン開発者であれば、ソフトウェアをCORBA（Common Object Request Broker Architecture）、Windows API、およびMFC（Microsoft Foundation Classライブラリ）で実行後にJavaとC#で保存していた1990年代をまだ覚えているかもしれません。しかし2000年代後半にコンピュータサイエンスの学位を取得した若手の場合、C++は教室で学んだだけだったり、まったく教わっていない場合があります。こうしたことから、C++のスキルを十分に備えた新しいチームを構築するには相当な時間と労力が必要になります。

ただしQtの場合、C++ソフトウェアの開発はMFCやCORBAの時代に比べるとずっとシンプルかつ体系的です。またC++の近代的かつ複雑な機能のすべてを知っておく必要もありません。現在のC++コミュニティはとてもアクティブで、言語そのものも進化し続けています。Linuxが組み込み市場やサーバ市場で人気を博すなか、プラットフォームに依存しないソフトウェアの重要性は高まる傾向にあり、C++のようなクロスプラットフォーム言語に有利な状況となっているのも事実です。

10年前にQMLが誕生したことで、近代的で滑らかに動くグラフィックUIの開発が可能になりました。C++と違い、QMLは強い型付けよりもシンプルなシンタックスを好む言語です。QMLはビジュアルデザインの構築に最適化されており、HTMLやJavaScriptの開発者にとっても無理のないエントリーポイントだと言えます。

HTML5アーキテクチャの場合、HTML5 UIはWebテクノロジーのみで開発したほうがメリットがあります。その際、バックエンドのすべてのインターフェースは、定義済みのサーバコールで取得しなければなりません。これはQtアーキテクチャでも可能ですが（図2参照）、その場合はごくシンプルなQt Quickアプリケーションであっても、モデルビューコンセプトの実行などでC++を使う必要があります。

QtとHTML5：典型的なハードウェア要件

以前の実験で、Qt／Qt QuickおよびHTML5（Angular）で同じアプリケーションを実行する比較を行ったところ、同じハードウェア（Raspberry Pi 3）であれば、Qt Quickアプリケーションのほうが滑らかに速く動くことが分かりました。また、ブラウザ構成に欠陥がある場合、OpenGLユニットとハードウェアの連動が十分に最適化されないことも明らかになりました。CPUもWebアーキテクチャではワークロード過多になることが判明しました。

一般に、組み込みデバイスでブラウザを使うと、ハードウェア要件は増えます。ブラウザとHTMLテクノロジーの組み合わせはネイティブアプリケーションと異なり、効率性のために最適化されていないからです。

以下は当社の過去のプロジェクトを基に、HTML5とQtという2つのテクノロジーの典型的なハードウェア要件をまとめたものです。

- HTML5アプリケーション、フルHD産業用パネル、Intel i3 2.2 GHz、2GB RAM
- HTML5アプリケーション、1024x768、Raspberry PI 2（1.2 GHzクアッドコア）、1GB RAM
- Qt Quickアプリケーション、1280x1024組み込みパネル、デュアルコア、1GB RAM
- Qt Quickアプリケーション、800x600組み込みパネル、i.MX 6デュアルコア、256MB RAM
- Qt C ++アプリケーション、480 x 320組み込みパネル、i.MX 28シングルコア、256MB RAM

図5は、典型的なハードウェア要件とユニット当たりコストの関係をまとめたものです。

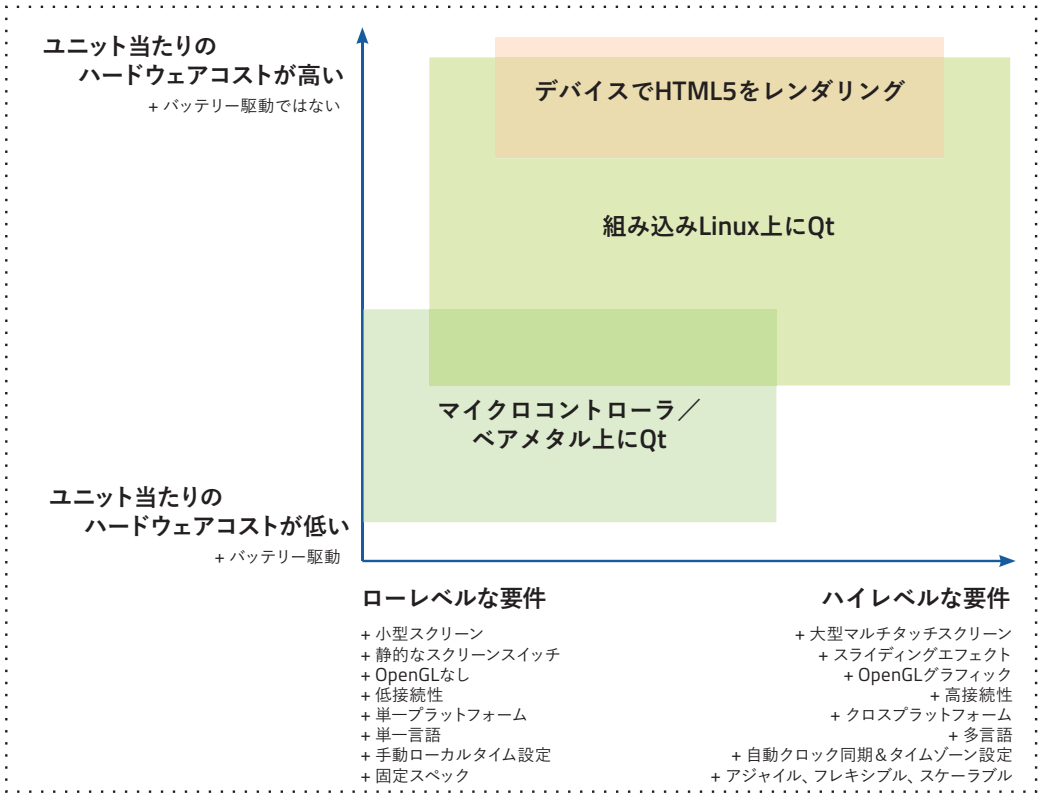

図5：「組み込みビジュアライゼーションマップ」

デバイスの生産数が多く、ハードウェアコストが低いほど、ソフトウェアスタックの効率性が重要になってきます。HTML5は通常、ハードウェア要件が高いため、生産数が多く、ユニット当たりのハードウェアコストが低いプロジェクトにはあまり向きません。このグラフでさらに注目するべき要素としては、開発コストと市場投入時間があります。いずれもテクノロジーを検討する際に等しく重要なポイントとなります。

HTML5とQtを組み合わせる – それぞれの良い点を生かす

タッチディスプレイにQt Quickアプリケーションを用い、HTML5アプリケーションをリモートコントロールとして使うメーカーが増えてきました。このような組み合わせには、以下のメリットがあります。

- アプリストアでインストールせずに、デバイスを遠隔制御できる。結果として展開作業を合理化し、異なるOS用のリリースを開発／管理する必要がなくなる。
- HTML5アプリケーションをデバイスに展開し、デバイスから配布できれば、インターネット接続なしで遠隔制御できる。
- HTML5アプリケーションと主なスマートフォン／ブラウザの互換性を確保できる。
- 近代的なシングルページアプリケーション（SPA）フレームワークをHTML5の実装（Angularなど）に使えるため、レスポンシブなブラウザアプリケーションの実装が容易になる（図5参照）。
- Qt Quickを使うことで、組み込みLinuxディスプレイにフルグラフィックやレスポンシブ性を取り入れられる。

デバイスのアプリケーションとブラウザのリモートコントロールアプリケーションは通常、ユースケースが異なります。そのため、UIも同じものではなく、それぞれのニーズに合わせて開発することになります。一方、ビジネスロジックを含むエンジン（図3参照）はどちらのアプリケーションにも同じテクノロジーニュートラルなインターフェースを提供するので（WebSocket経由のJSONなど）、1度の開発で済みます。

新たなHTML5アプリケーションの開発に取り掛かる際はぜひ、Qtテクノロジーの最新バージョンである「Qt WebGL Streaming」と「Qt for WebAssembly」も事前に検討してください。いずれのテクノロジーも、既存のQtアプリケーションをWebブラウザで視覚化することができます[1]。

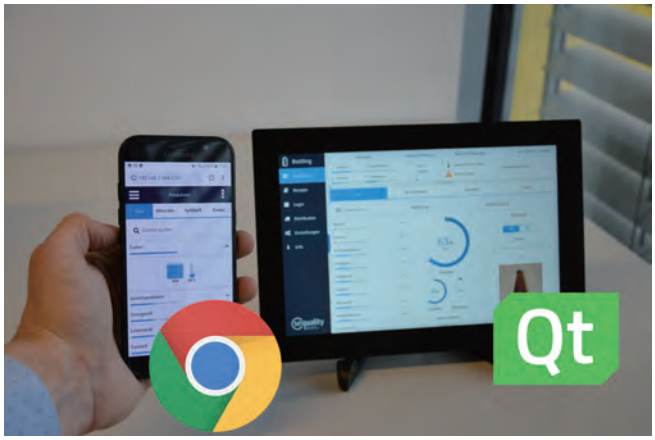

図6：スマートフォンブラウザのHTML5アプリケーションと組み込みLinuxタッチディスプレイのQt Quickアプリケーション。

▪Qt WebGL Streaming：NCクライアントと同様、QMLアプリケーションをブラウザに配信する際に、追加のコードをHTMLに転換する必要がありません。ただし、アプリケーション自体は組み込みデバイス上で実行します。このアプローチは、ヘッドレスデバイスにも適しています。アプリケーションを複数のユーザーで使いたい場合は、アプリケーションのインスタンスを複数実行し、互いに同期させなければなりません。従って、エンジンとビュワーアプリケーションが別々のスプリットアーキテクチャが必要です。

▪Qt for WebAssembly：WebAssemblyは一種の仮想マシンで、WebAssembly用にコンパイルされたQtアプリケーションをブラウザ内で直接実行します。なお、これはスプリットアーキテクチャとなり（図3参照）、WebAssemblyアプリケーション自体は組み込みデバイスではなく、ユーザーのブラウザで実行されます。また、エンジンとのデータ交換には追加的な通信チャンネル（WebSocket+JSONなど）が必要です。

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

ソフトウェアスタックの比較

HTML5アプリケーションまたはQtアプリケーションを実行するシステムは、複数のソフトウェアレイヤーとその下のハードウェアレイヤーで構成されます。ソフトウェアレイヤーはおおむね、選択したアーキテクチャがもたらす依存性によって決まります。ソフトウェアレイヤーは、以下の2つの方法で設定可能です。(1) 既存のLinuxディストリビューションをニーズに合わせて調整する、または (2) Linuxビルドシステムを使ってカスタムLinuxシステムを構築する。図7はカスタムRaspbian (1) のソフトウェアレイヤーで、図8と9はYoctoビルドシステムを用いています。

既製のGoogle Chromeブラウザを用いたHTML5アプリケーション

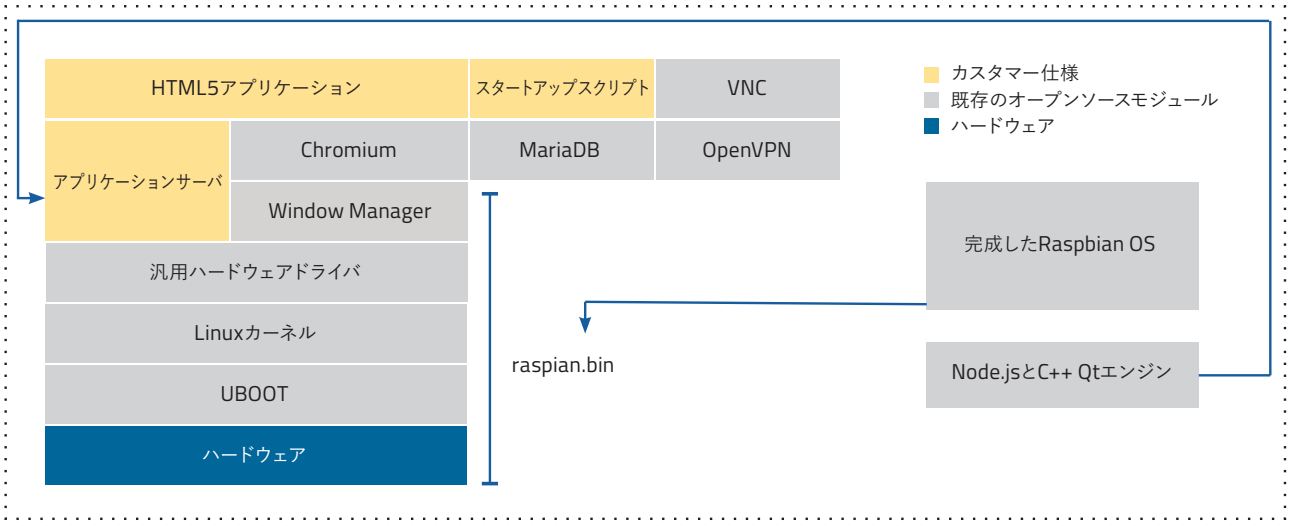

図7：標準的なChromiumブラウザとカスタムRaspbian OSの組み合わせ

図7は、典型的なHTML5アーキテクチャ（図1参照）の「sequality Bottling HTML5」デモアプリケーションの構成をまとめたものです。灰色のレイヤーは既製のオープンソースソフトウェアで、黄色のレイヤーはカスタマー仕様のソフトウェアモジュールです。

製品によっては複数のディスプレイを表示する必要がありますが、たいていは単一のメインアプリケーションのみ表示します。単一のメインアプリケーションだけであれば、ウィンドウマネージャはいりません。ただし、多くの標準的なブラウザはXサーバ環境でのみ実行可能なため、このソフトウェアレイヤーもやはりシステムに組み込む必要があります。

安全なキオスクモードを実行するには？

ウィンドウマネージャが必須であるということは、複数の不要な機能も付加されることを意味します。たとえばアクティブウィンドウの切り替えを行ったり、場合によっては実行中のアプリケーションを閉じたりする「Alt+Tab」がそうです。しかしながら、Linuxを「キオスクモード」で実行すれば、これを回避できます。

Yoctoを用いた Qt Quickアプリケーション

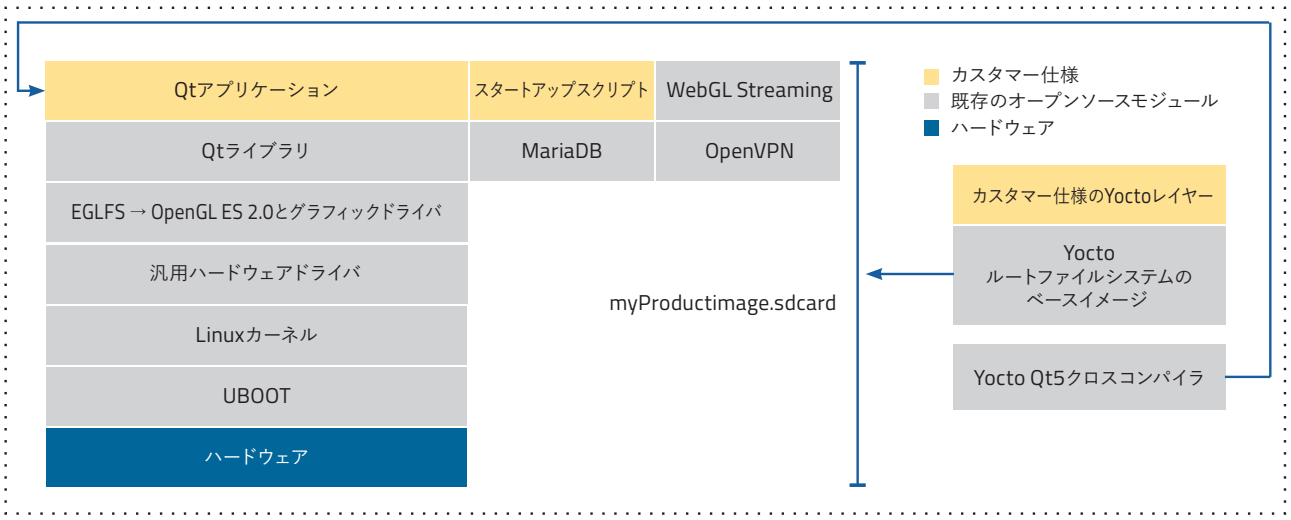

図8：ウィンドウマネージャのないQtアプリケーションのソフトウェアレイヤー構成

Qtアプリケーションは、Xサーバの有無に関わらずOpenGLグラフィックインターフェース経由で直接実行できます（リンクページのEGLFSのセクションを参照）。図8は、図2または3のQt Quickアプリケーションの典型的なソフトウェアレイヤー構成をまとめたものです。多くの場合、メモリ消費量の多いXサーバの代わりにEGLFSインターフェースを使ってQtアプリケーションのレンダリングを実行することができます。このアプローチでは、安全なキオスクモードを実行しながら、ブートタイムを改善することが可能です。

Yoctoを使うことで、製品のルートファイルシステム（図8の「myProductImage.sdcard」）を繰り返し正確に生成できます。YoctoはQt5クロスコンパイラも提供するので、これをLinuxデスクトップ環境にインストールし、ターゲットシステム用にQtアプリケーションをコンパイルできます。

QtWebViewを用いたHTML5アプリケーション

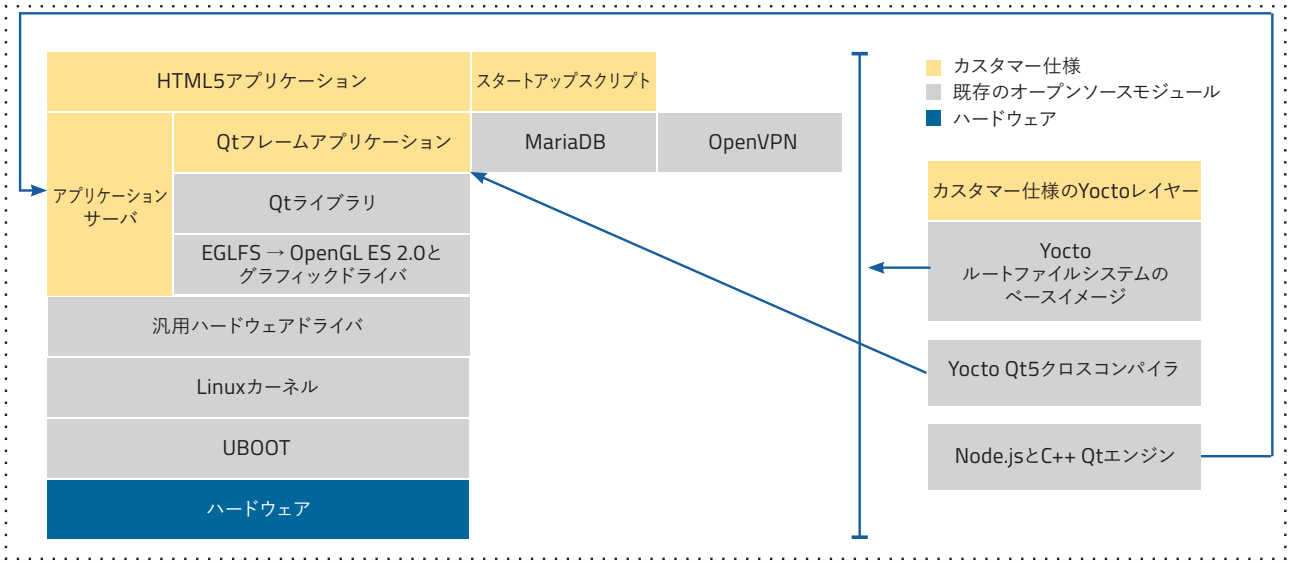

図9：ソフトウェアレイヤー構成とQtWebView

HTML5でソフトウェアを実装すると決めたら、キオスクアプリケーション、組み込みデバイス、または製品用に最適なブラウザを実装する方法を考えるのが合理的です。Google ChromeやMozilla Firefoxといった標準的なブラウザはプラグインインターフェースを提供しますが、カスタム拡張（または制約）を追加する実用的な方法はありません。

Qtフレームワークが提供するQtWebViewは、Chromiumベースのブラウザコンポーネントで、製品のニーズに合わせてカスタマイズすることができます。

メリット：

- Xサーバが不要：QtアプリケーションはEGLFSグラフィックインターフェースを使ってウィンドウを表示できます。そのため、ソフトウェアスタックを合理化できるほか、ユーザーがキーボードコマンドで新たにウィンドウを開いたりウィンドウを操作したりできるセキュリティホールやバックドアからソフトウェアスタックを保護することができます。キオスクソフトウェアを使う必要はありません。
- OpenGL対応：Qtフレームワークを使い、Qt Yoctoレイヤーと統合した場合、OpenGLに対応します。
- スクリーンセーバーをコントロール：非アクティブディスプレイの輝度を下げて製品の効率性を高めたい場合、Qtフレームワークアプリケーションと QtWebViewの組み合わせなら、わずか数行のコードで実現することができます。
- 仮想キーボードの統合：　タッチディスプレイのHTML5アプリケーションでテキスト入力を行うには仮想キーボードが必要です。Qtフレームワークは既製の仮想キーボードコンポーネントを提供しており、QtWebViewと連動させることができます。さらにQtならカスタム仮想キーボードの実装も可能です。
- Qtフレームワークアプリケーションはデバイスのハードウェアスイッチ／ボタンとダイレクトに接続できます。そのため、ハードウェアイベントをサーバ経由で通知することも、Webブラウザソフトウェアにダイレクトに接続することもできます。
- 安定したリリースサイクル：Qtフレームワークは定期的なリリースサイクルで開発されており、極めて安定しています。従って、内蔵のWebViewコンポーネントも今後数年間にわたって定期的に更新されます。
- ブラウザのGUIを100％コントロール：GUIエレメントの中には、バック／ホームボタンやローディングバー、マウスカーソルなど、セキュリティ上の理由から要不要を判断するものもあります。標準的なブラウザでは、この調整は非常に面倒です。しかし、QtWebViewなら、これらの調整作業や、右マウスボタンやキーボードショートカットのコントロールが比較的容易です。
- カスタマー仕様のエラー管理とロギング：HTML5アプリケーションでエラーが発生した場合、自動ロギングして後から分析したり、担当者に転送したりする必要があります。しかしこれは標準的なブラウザでは困難です。QtWebViewフレームワークアプリケーションなら、たとえばJavaScriptでエンドレスループロギングを行い、カスタマーの固有ディスプレイとして視覚化できます。また、リンクが壊れた、サーバへの接続が切れたといった場合にも、重要な画面をユーザーに提供できます。
- セキュリティと安全性：　ブラウザは製品の必須機能のみに継続的に縮小したほうが、セキュリティが高まり、攻撃への脆弱性が改善されます。

図10：HTML5 sequality- 瓶詰め工場向け組み込みLinuxターミナルのQtWebViewのデモアプリケーション。

Yocto及びHTML5リモートコントロールを用いたQt Quickアプリケーション

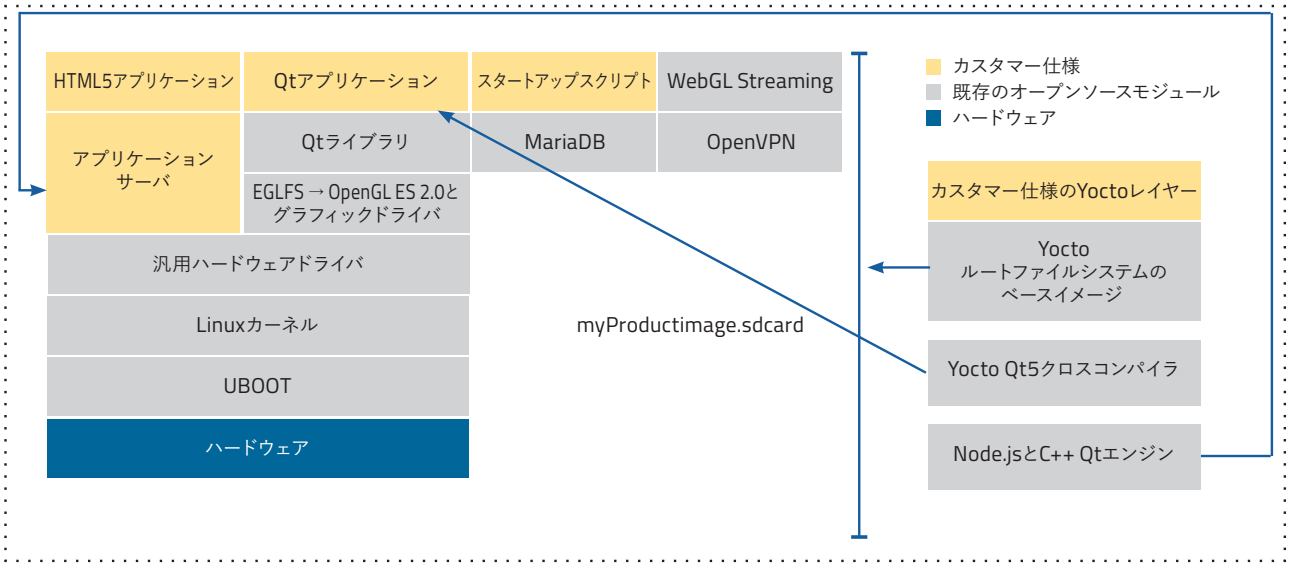

図11：リモート操作のためのHTML5を用いたQtアプリケーションのソフトウェアレイヤー構成

場合によっては、ネイティブQtアプリケーションに加えて、HTML5を使ったリモートコントロールアプリケーションをデバイスに実装するのもおもしろい試みです。通常、スマートフォンやタブレットといった外部デバイスは、スマートフォン／タブレットのWebブラウザを使ってリモートコントロールとして機能し、デバイスのHTML5アプリケーションを実行します。その場合、HTML5アプリケーションには外部デバイスとそのブラウザからしかアクセスできません。またデバイス上の情報は、Qtアプリケーション経由でしか表示できません。QtとHTML5のいずれのアプリケーションも、アプリケーションサーバを使ってビジネスロジックを共有します。

まとめ

HTML5もQtも、製品開発には素晴らしい選択肢です。ただし、開発プロジェクトにおいて万能薬は存在しません。環境によって、明らかにQtが適している場合もあれば、HTML5が適している場合もあります。ふさわしい技術を選択するに当たっては、アーキテクチャ全体、つまりフルスタックをベースに検討しなければなりません。また、ライブラリ間の依存性とその安定性を検討することも欠かせません。さらに、ソフトウェアスタック全体の安定性と機能性が、ライフサイクルを通した製品の可用性および拡張性を左右する点にも配慮が必要です。

HTML5アプローチを取るなら、Qtライブラリのモジュールの利用を検討することが推奨されます。特に、産業用オートメーションやバックエンドプロトコル（Modbus、OPC-UAなど）関連のモジュールが重要です。Qt WebViewは製品固有のブラウザを最も合理的に実装できるアプローチであり、安定したアップデートサイクルとカスタムハードウェアサポートという強みもあります。

どのアプローチを選ぶにせよ、オープンでスケーラブルなアーキテクチャを目標に掲げたプランを策定することが大切です。テクノロジーニュートラルなビジネスロジックインターフェースを備えたスケーラブルなアーキテクチャで製品開発を行えば、完成したハイブリッドシステムの実装もずっと簡単です。このアプローチなら、状況に合わせて理想的なテクノロジーを自由に活用でき、複雑なビジネスロジックをそのつど実装する手間もいりません。

出　典

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/jp/whitepaper-qt-vs-html5-japanese-ver

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/

筆者について

ステファン・ランドーファー（Stefan Larndorfer）はSequality software engineeringのCEOで、自動車、産業オートメーション、医用工学などさまざまな領域で多数のプロジェクト管理に携わっています。現在はチームと共に、組み込みLinuxを基盤とした革新的なソフトウェアソリューションの開発を行っています。また、ハーゲンベルグ応用科学大学においてC++Qtの講師を務めているほか、オープンソース開発を推進し、多種多様な研究プロジェクトにも参加しています。

Stefan.Larndorfer@sequality.at

Sequalityについて

Sequality software engineeringはオーストリアを本拠とするソフトウェアコンサルティング会社です。産業アプリケーションやタッチディスプレイ用UI、組み込みソフトウェアアプリケーションなどの各分野において、クライアント企業のソリューション開発をサポートしています。

Sequalityがアプリケーション開発で重視しているのは、ユーザビリティです。私たちはユーザビリティエンジニアおよびUXデザイナーと協働して、ユーザーフレンドリーな機能や優れたルックを備えた最先端のUIテクノロジーアプリケーション開発を推し進め、シームレスなUI体験を提供しています。

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