The Qt Company

WHITEPAPER

IoTの構築：Qtを使ったアプローチ

目次

IoTの構築：Qtを使ったアプローチ

IoT（モノのインターネット）という言葉は、今やあらゆるところで目にします。聴衆に関連性を感じてもらうための最新のバズワードといえるでしょう。SFの世界では、IoTは日常品のさまざまなところに組み込まれた極小サイズのコンピュータとして登場します。しかし、多くの開発者にとって、IoTは「接続された組み込みシステム」というお馴染みのコンポーネントを、より洗練された新しい名前で呼び換えているに過ぎません。では、開発者はIoT的なデバイスを数十年間にわたって作り続けてきたのでしょうか？　答えはイエスでもあり、ノーでもあります。

IoTデバイスを定義する場合、特に重要なポイントとして、組み込みシステムであること、多くの場合はモバイルであり、M2M（マシン・トゥー・マシン）を用いている、つまり機械同士の通信を実現するガジェットであることが挙げられます。もちろん、これらの属性は多くの組み込みデバイスにも当てはまります。ただし、IoTデバイスは常時通信により、スマートに稼働するために必要な情報を直ちに取得できるという特徴も備えています。このように、日常品に基本的な情報を送ることができ、通信機能も備わっているIoTデバイスは、多種多様なテクノロジー製品の開発を可能にしてくれます。たとえば、センサー内蔵の生体認証ウェア、セルフスケジュール機能を備えた宅配ドローン、オートモニタリング機能を持った住宅、鮮度情報をもつ生鮮食品、自動パーキングメーター、自己診断型農作物など、数々の技術を実現することができます。

こうしたIoTのビジョンを現実のものにするには、コンピュータの小型化および高度化、さらに接続性の向上を継続していかなければなりません。そのためにはハードウェアの変革が必要だという点は周知の事実のようですが、ソフトウェアの面でも大きな変革が必要になるという点については、あまり議論が進んでいません。日常にあふれる様々なモノに情報を持たせるのと同時に、H2M（ヒューマン・トゥー・マシン、人と機械）およびM2Mという2種類のコミュニケーションをより自然に、直感的に実行できるようにするには、膨大な量の複雑なソフトウェアが必要になります。そのため、必然的に、IoTソフトウェアの開発にも多くの要件が加わることになります。

IoTのソフトウェア要件

インテリジェントなコネクテッドシステムの開発に活用できるテクノロジーには多くの種類があるため、何から手をつければいいのか判断しづらい場合もあるでしょう。そのような場合、まずは理想的なソフトウェアフレームワークの構成要素について（あるいは基本的な要件について）考えてみるのが良いでしょう。

1) 高性能

高度な計算能力がなければ、「スマートデバイス」は実現できません。必要なのは、タスクに見合ったソフトウェアツールキットです。

2) 最適性

プロセッサと基盤を小型化しても、無限にリソースが得られるわけではありません。必要なのは、高性能かつ効率性に優れたソフトウェアです。

3) 接続性

処理するべき通信形態はさまざまなので、柔軟な接続オプションを用意して、ネットワークのエッジにあるスマートデバイスをクラウドにつなぐ、センサーネットからデータを取得する、といった機能を実現する必要があります。また、多様なプロトコルやスタック、ワイヤレステクノロジーに容易に対応するツールを選ぶことも重要です。

4) RAD（高速アプリケーション開発）

フェイルファスト（速く失敗する）を実践して製品を進化させ、IoTのスピーディなライフサイクルに対応するには、ソフトウェアの開発も高い信頼性を維持しながらスピーディに行わなければなりません。近代的なツールとIDE（統合開発環境）を基盤とする、可能な限りシンプルな開発プロセスが必要です。

5) クロスプラットフォーム

デスクトップからクラウドサーバまで、あるいはヘッドレスセンサーから組み込みUI（ユーザーインターフェース）まで、可能な限り多くのデバイスでコードベースを共有できるのが理想です。従って、開発ツールはポーティング作業を最小限にできるものを選ぶことが不可欠です。

6) ユーザーインターフェース

多くのIoTデバイスはUIを必要としません。しかしながら、コネクテッドガジェットにはUIが必要なものが多々あります。画面の有無で開発ツールチェーンを分けるのは理想的とは言えません。すべての環境に適切に対応できる開発ツールが見つかればベストです。

7) シェア

どんなものでも一から作り直すのは面倒です。従って、仲間から学ぶことができる、オープンで活発、かつグローバルな開発コミュニティも必要でしょう。

8) 安全性

ハッカーはどこにでもいます。ハッキング攻撃を防ぐことができ、なおかつ回復力の高いデバイスを開発しなければなりません。

9) 信頼性

「スマート」デバイスがいくつあったところで、動きが不安定で、頻繁に再起動が必要なようでは意味がありません。スマートデバイスには高い信頼性が必要です。言語やツールはすべて、信頼性に優れたソフトウェアを開発できるべきではありますが、言語やツールの選択に際しては、外部の評価や認定の有無を判断基準とするのが良いでしょう。これは必ずしもどの業界にも当てはまるものではないかもしれませんが、少なくともIoT業界では、これらの判断基準が重要です。

10) 安定性

ソフトウェアは継続的に変化するものです。デバイスの機能が固定されたソフトウェアモデルが出荷時点ですでに時代遅れになりつつある、ということも珍しくありません。従って、ソフトウェアを使いながら更新できること、デバイスの開発に使ったソフトウェアツールが（リフレッシュ後でも）安定性に優れていることが不可欠です。

パラメータを絞り込んだところで、続けて代表的な組み込み言語とフレームワークを紹介します。
さらにコンビネーションごとの構成要素のスコアをつけてみましょう。

	C	C++/STL	C++/Boost	C++/Qt	C#/.NET	Java/ Android	HTML5/ Cordova
高性能	–	✓	✓	✓	✓	✓	–
最適性	✓	✓	✓	✓	–	–	–
接続性	–	–	✓	✓	✓	✓	✓
RAD	–	–	–	✓	✓	✓	✓
クロス プラットフォーム	✓	✓	✓	✓	–	–	✓
ユーザー インターフェース	–	–	–	✓	✓	✓	✓
シェア	–	✓	✓	✓	–	✓	✓
安全性	–	✓	✓	✓	✓	–	–
信頼性	✓	✓	✓	✓	–	–	–
安定性	✓	✓	✓	✓	✓	✓	–

C

C言語はデスクトップ開発では廃れてしまいましたが、組み込みの分野では依然として人気があります。ただしC言語の世界には、最先端の技術が存在しないという大きな問題があります。IDEやRAD、フレームワーク、コミュニティを基盤とした開発は、C言語の世界では進んでいません。

スコア:4/10

C#/.NET

C#および.NETには多くの長所があるものの、C++に比べると効率性やコンパクトさに劣り、最適化も図られていません。また、Monoをとってみても、クロスプラットフォームの互換性に欠ける問題があります。また、コミュニティは活発ですが、オープンソースほど熱心とは言えません。

スコア:6/10

HTML5/Cordova

HTML5(またはJavaScript + HTML + CSS)とApache Cordovaは、クロスプラットフォームの場合や開発オプションとしては高い人気がありますが、多くのIoTシステムには大きすぎ、あくまで次善策といったところです。しかもHTML5フレームワークの大半は流動的です。新機能を頻繁に追加するような場合は有益ですが、安定したコードベースを構築して、アップデートしながら保守を行っていくような場合には望ましくありません。

スコア:5/10

Java/Android

Androidは優れたソフトウェアですが、開発環境が重いという欠点があります。とはいえ一番の問題はクロスプラットフォームではない点です。Androidのアプリケーションは Androidデバイスでしか動作しないため、コードの再利用が制限されることになります。

スコア:6/10

C++/STL

C++はパワフルかつ最適な言語であり、組み込み言語としては今でもベストと言えるでしょう。C++にSTLを組み合わせれば、その表現力をさらに高めることができますが、接続性についてはあまり効果が期待できません。SOCKSやBluetooth、Wifiといった多様なライブラリを利用することもできますが、代わりに万能なソリューションとしてのメリットは得られなくなります。またC++/STLは、RADやUI開発に不向きである点もネックです。

スコア：7/10

C++/Boost

STLに代わってBoostをC++に組み合わせることで、接続性を改善できます。ただしBoostは単なるライブラリに過ぎないので、開発環境という意味ではデメリットが残されます。また、標準的なUIツールがないという欠点もあります。

Score: 8/10

C++/Qt

最後に、C++とQtを見てみましょう。QtはC++のベアメタル性能を最大限に活用しながら、同時に、アプリケーションやUIコンポーネントのRADも実現します。また、UIコンポーネントをクロスプラットフォーム開発でき、完全なIDE環境と活発なコミュニティのおかげで、その他の要件についても効果を発揮できます。

スコア：10/10

IoTにおける課題

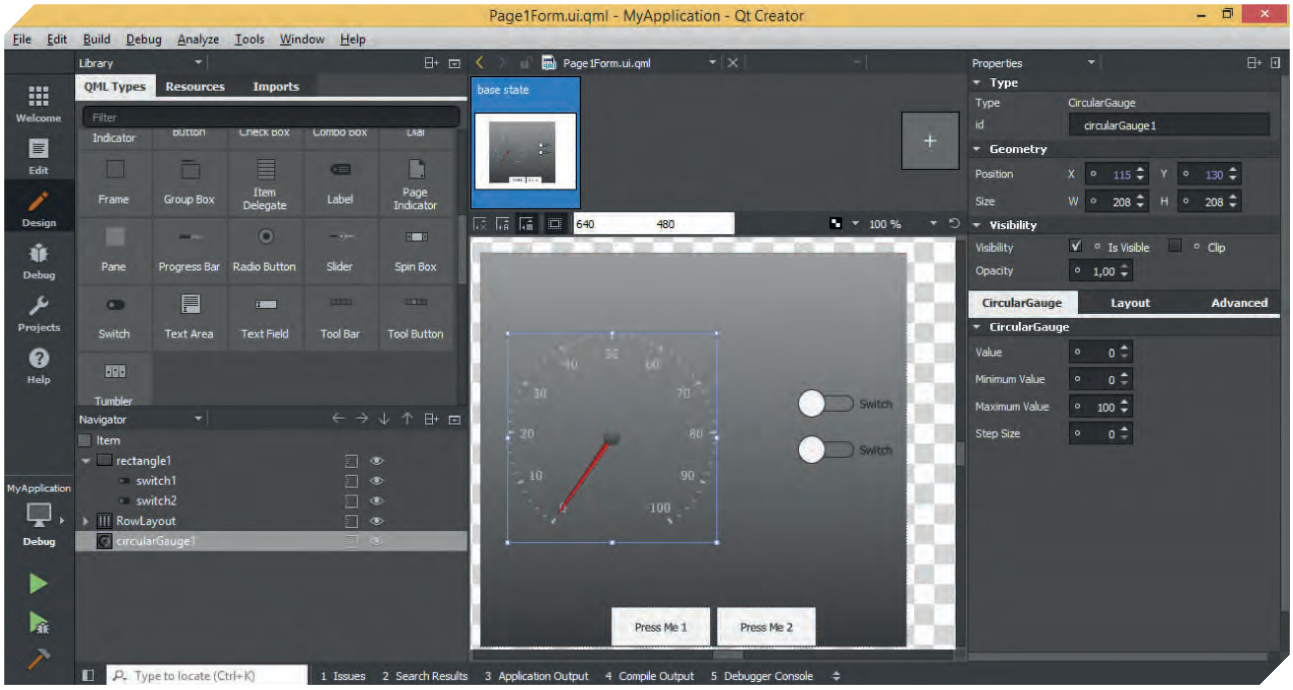
この比較でみたように、Qtがベストな選択肢といえるでしょう。ただし、何が最良のIoTツールキットかを決めるのは、コードの行数や、エコシステムを構成する開発者の人数ばかりではありません。重要なのは、そのツールキットをいかにスピーディかつ効果的に活用できるかという点です。次のセクションでは、IoTのさまざまな課題について、また、Qtがそれらの課題をいかに克服できるかについて詳細に見ていきましょう。

組み込み開発

組み込み開発の大部分は、大昔から進化していないようです。リポジトリプラグ、オートコンプリート、シンタックスカラーリング、統合デバッグ、プラグインホスティング、クラウドコネクテッドIDEといった近代的な開発環境と、ブラックスクリーン上で緑色に点滅するカーソルを動かす開発環境では、オートスタートできる電気自動車のテスラと、手動エンジンスタートが必要な大昔のT型フォードほどの違いがあります。ターミナルセッションやコンソールスクリプト、vi/emacsなどは今も現役ですが、GUIに至っては誕生からすでに30年以上もの時が経っています。

Qtはここでも大きなアドバンテージを有しています。Qt Creator IDEはあらゆるプラットフォームに対応でき、あらゆる拡張機能、プラグイン、ショートカット、タイムセーバーをサポートし、あらゆるターゲットにクロスコンパイルできます。また、コードのデバッグ、プロファイリング、分析、デザインに必要なツールも完備しています。対応する環境の互換性も確保されるため、パソコンでラピッドプロトタイピングを行い、万全の状態を確認したところで、組み込みターゲットに移植することが可能です。Qtは単なるUI開発ツールだと言う向きもありますが、実際には多種多様なQtライブラリがあり、たとえばヘッドレスIoT開発なら、国際化、ストリング、スレッド、XML、JSONパーシング、データベース、ソケット、Bluetooth、センサー、NFC、イベント管理などのライブラリを提供します。

現代の開発者の大部分は、他のデバイスとコードを共有しない固有の一製品の開発作業に専念できるほど余裕がありません。だからこそ、UI開発が得意な組み込みツールが重要なのです。そのような組み込みツールがあれば、(スクリーンの有無を問わず)複数のデバイスをサポートでき、デスクトップインタフェースとモバイルガジェットで主要な通信ライブラリを共有でき、IoT開発者エコシステム内で別のコンポーネントを担当する仲間とコードを共有できます。またQtなら、UIデバイス以外のターゲット向けに最低限必要なコードベースを開発し、そこにデスクトップやモバイル、タブレット用の高度なインターフェースを重ねることも可能です。



Qt Creator IDE のQt Quick Designerでは、使い勝手の良いドラッグ&ドロップ機能を使って、ユーザーインターフェースのビジュアルプロトタイプを作成することが可能です。

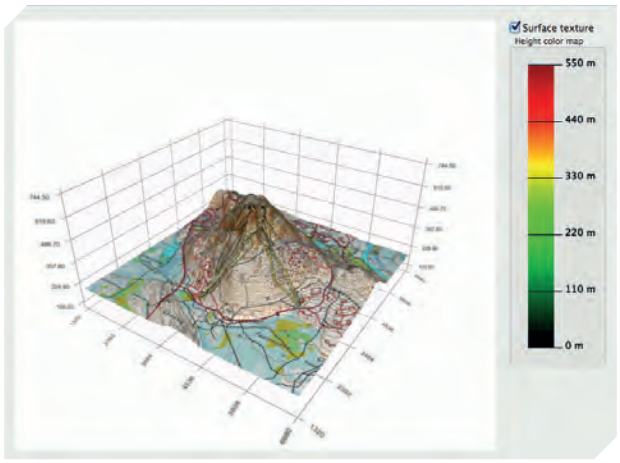
ビッグデータ

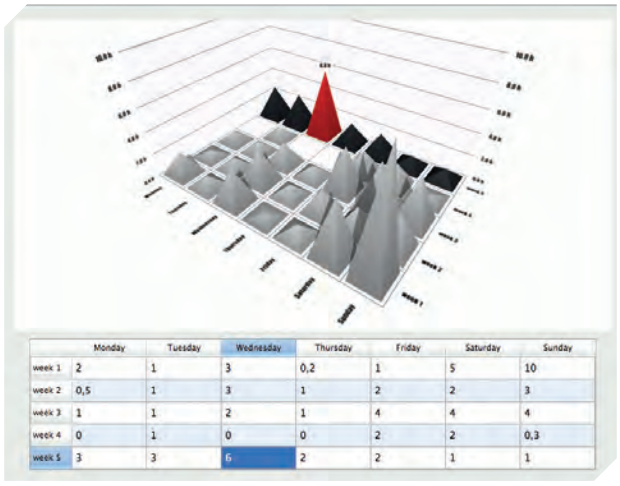
ビッグデータはIoTに匹敵する注目を集めており、IoTの分野では重要な概念の1つです。数多のデバイスで継続的に収集される膨大なデータからは、大きな「知」を生み出すことができます。中には、ビッグデータが関係してくるのはクラウドの分野だけだと思っている方もいるかもしれません。しかし実際には、クラウドと規模こそ異なれ、ビッグデータの管理はエンドポイントにおいてもやはり必要です。IoT組み込みシステムが扱うデータはエクサバイト規模ではないかもしれませんが、ギガバイト規模のデータの収集、保存、フィルター、処理は必要になる場合もあるのです。

この規模のワークロードを扱うには、メモリ消費を抑えながらデータを操作し、効率的にフォーマット変換を行って、永続的にデータを保存しなければなりません。C++では、プログラマーがデータ保存フォーマットを直接管理するので、外部メタデータやガベージのコレクションは不要です。つまりC++はメモリを意識した言語であり、Qtをはじめとする C++ベースの開発環境はいずれも望ましい選択肢だと言うことができます。ただしフォーマット変換に当たっては、MQTT、JSON、生のバイナリデータのセンサー間での送受信を、できるだけバッテリを消費しない方法で行う必要があります。これはどの選択肢でも可能なわけではありません。Qtは、これに対応する専用ライブラリと高速性を有しています。

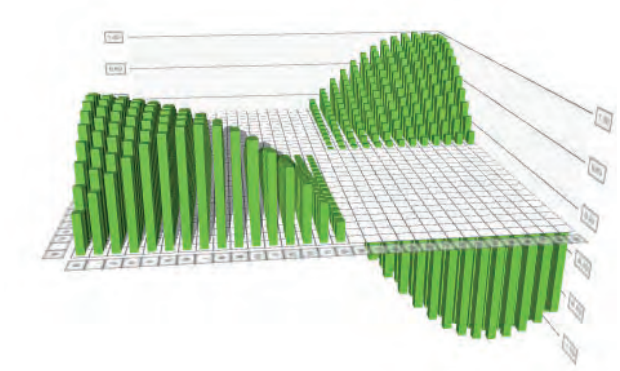
保存や参照が必要なデータをデータベースに書き込むためのAPIも欠かせません。
QtはSQL（MySQL、Postgres、SQLiteなど）のインターフェースを提供するので、収集したデータを簡単に保存・処理できます。

ビッグデータについてはもう1つ、検討するべき側面があります。データの可視化の問題です。データの可視化はエンドユーザーにはほとんど不要ですが、生成されたデータセットを分析・解釈するデータサイエンティストにとっては欠かせないプロセスです。Qtは可視化のための専用モジュールを提供するので、2Dまたは3Dチャートを作成し、さまざまな方法でデータをより詳しく表示できます。





	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
week 1	2	1	3	0,2	1	5	10
week 2	0,5	1	3	1	2	2	3
week 3	1	1	2	1	4	4	4
week 4	0	1	0	0	2	2	0,3
week 5	3	3	6	2	2	1	1



コンシューマーエクスペリエンス

一般に、IoTデバイスはスタンドアロンではありません。中にはタッチスクリーン（ホームコントロールパネルなど）と、ドットアドレス指定可能な大型グラフィックを使ったIoTデバイスもありますが、少数派です。従ってユーザーは、デバイスと通信したり、デバイスの設定を行ったり、データのダウンロードやステータスの表示・変更を行ったりする手段が必要になります。多くのIoTデバイスは、シンプルな点滅式LEDとボタンのインターフェースでUI要件に応えています。人とデバイスのインタラクションはコンシューマーエクスペリエンスの根幹を成す部分であることから、IoTデバイスの成功の鍵を握る非常に重要なポイントとなります。また多くの場合、IoTデバイスとユーザーのインターフェースはデスクトップやモバイル、タブレットのUIを介して間接的に行われます。

Qtは当初からユーザーインターフェースが開発できるように設計されています。これはIoTデバイスの開発に欠かせない機能です。Qtなら、シンプルなもの、複雑なもの、クラシックなもの、近代的なもの、標準化されたもの、カスタマイズされたものなど、あらゆるUIを作成することが可能です。しかも、Qtで作ったコンポーネントはすべてクロスプラットフォームです。ユーザーがMicrosoft派であれ、Appleマニアであれ、Linuxファンであれ、すべてのデスクトップでIoTインターフェースを運用できるのです。もちろん、iPhoneとAndroidについても同様です。1つのプラットフォームだけをサポートするようなアプローチは、ユーザーに不満を与えるだけです。WindowsやTizen、Blackberryなどのスマートフォンを使っている7～8%の少数派なら尚更、このアプローチには不満を覚えるはずです。Qtなら、単一のコードベースを用いることで、OSや携帯電話の種類にかかわらず全ユーザーをシンプルにサポートできます。

単一のコードベースについてさらに触れておくと、Qtのもう1つの大きな利点として、組み込みデバイスのファームウェアと制御用アプリケーションでソフトウェアを共有できることが挙げられます。
古き悪しき時代には、RADに複数の言語やデータベースが使われており、たとえばデスクトップをVBやDelphiで、組み込みをCで、それぞれ開発する必要がありました。つまり、1つのコードで2つのインスタンス（2つの言語、2つのAPI、2つのテストスイート）が存在することになり、バグを隠す方法も2種類になるわけです。Qtなら、1つのデータパーシングモジュールやコンフィギュレーション管理ライブラリ、プロトコルスタックをデザイン、開発、テストした後、ソフトウェアを運用するすべてのターゲットで共有することができます。

拡張性

新製品を開発するたびに、ソフトウェアを書き直すほどの余裕は開発チームにはありません。コストがかかるだけではなく、新製品の開発スピードを著しく低下させることにもなります。それにもかかわらず、数年おきにソフトウェアを廃棄し、一から作り直しているのが実情のようです。ソフトウェアの作り直しは大抵、想定外の要件変更のほか、製品設計時の見通しが甘かったために発生します。最悪なのは、システムに不可欠なハードウェアやソフトウェアの依存性に陥り、製品のサポートが終了した場合や製品ロードマップに支障をきたしたために、書き直しを余儀なくされるケースです。

いずれの言語やフレームワークを選んだところで、いつまでも有効に使い続けることができる万全の解決策というものは存在しません。従って、経験を基に将来においても有効な設計を目指し、拡張性と適応性に優れ、修正がしやすいソフトウェアを作ることが重要です。Qtは以下のようなかたちで、変化し続ける状況にも容易に適応できるよう、製品開発をサポートします。

新しい組み込みハードウェア

Qtは、大規模かつオープンな開発コミュニティと組み込み分野における高評価により、新しいボードに真っ先に移植されるフレームワークとなるはずです。ハードウェアの変更や拡張が必要になった時にも、自力で移植を行う手間はかかりません。

新しいアプリケーション環境

新たなスマートフォンやUIパラダイム、Linuxディストリビューションが誕生した場合には、活発なQtコミュニティがフレームワークの最適性を維持し、デバイスに適した新たなプラットフォームを開発して、ユーザーとのインターフェースをサポートします。Qtは確かなクロスプラットフォーム対応により、早期の移植を実現します。

新しいセンサー

Qt Sensor API には多数のセンサータイプが存在します。新しいセンサーがこれらのセンサータイプのいずれかに属するものなら、シンプルなプラグインを開発してこの抽象化レイヤーを活用することができます。

新しいプロトコル

Qtにはさまざまなネットワーキング＆コネクティビティクラスがあり、新たなワイヤレステクノロジーに既存のソケット通信を容易に用いることが可能です。新しいプロトコルは通常、実行に新しいクラスを必要としますが、多数ある既存のクラスを利用することで、プロセスを多少なりとも簡素化することができます。

新しいバックエンドコネクション

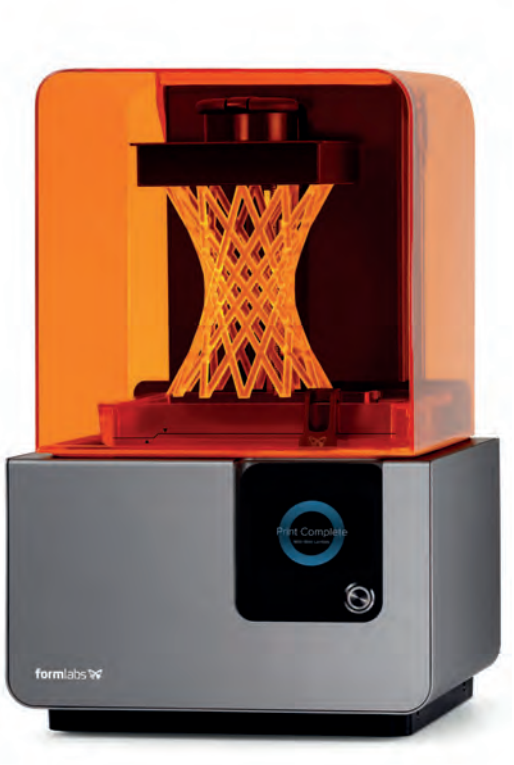
IoT UIでクラウドサービスへのアクセスが必要になったとします。そのような場合も、QtはRESTfulおよびSOAPインターフェース、XMLおよびJSONパーシング、ならびにクラウドAPIを提供するので、いつでもAWSやAzureにアクセスすることが可能です。

プラグインサポート

機能をダイナミックに追加できるIoTアーキテクチャーが必要なら、プラグインの追加を検討するという手もあります。プラグインなら、開発者やそのコミュニティはベースコードの書き換えを行わなくても、製品に新しい画期的な機能を加えることが可能です。Qtには、提供する各種機能向けのさまざまな標準プラグインがあります。特にIoT開発者向けとしては、固有のニーズに最適なプラグイン抽象化レイヤーを簡単に開発できるベースプラグインアーキテクチャも提供します。また、プラグインのように高度なモジュラーインターフェースは必要ないという場合には、多種多様なサードパーティライブラリをQtと組み合わせて利用することもできます。たとえば、C/C++ライブラリのほかにも、Qt独自のライブラリが多数あります。

とはいえ、拡張性を高めるために世界中のあらゆるツールを入手したところで、それらを活用できなければ意味がありません。

Qtであれ、他のフレームワークであれ、既存のモジュールを手持ちのシステムに統合する作業はゼロから始めなければなりません。ただし、活用しているプロセスが役立つこともあります。たとえばアジャイル開発は、拡張性に優れたシステムの構築に適しています。アジャイル開発が、スクラム毎にシステムを拡張していくモデルだからです。いずれの開発プロセスを取るにせよ、容易にユニットテストが行えるようにモジュール化および抽象化して記述する、レイヤー間の結合が少なくて済むように記述する、各タッチポイントと依存性を可能な限り分離するといったアプローチを心がけることが、IoTソフトウェアに柔軟性を持たせ、継続的に活用していく上で重要だと言えるでしょう。



まとめ

IoTの世界では、よりスマートで接続性に優れたデバイスを開発するために、今まで以上にソフトウェアが非常に重要になっています。ソフトウェアフレームワークの選択基準が、本書の定義と同じという方もいれば、違うという方もいるでしょう。いずれにしても、高性能で最適性と接続性に優れ、RADが可能、クロスプラットフォーム対応、ユーザーインターフェースと共有機能を持つIoTデバイス開発を目指すなら、Qtを検討する価値があります。しかし、これらの基本的な属性ばかりが、ソフトウェアの決め手となるわけではありません。

多くのIoTソフトウェアにとって、組み込み開発が容易である、ビッグデータの取り扱いがスムーズである、コンシューマーエクスペリエンスに優れている、ソフトウェアアーキテクチャが常に進化しているといったQtのその他の特長もプラスに働きます。

IoTソフトウェアで何を行うにせよ、大切なのは、長期にわたって活用できるフレームワークを構築することです。選択は是非とも慎重に行ってください。

お問い合わせ：
The Qt Company　日本オフィス
〒100-0005
東京都千代田区丸の内3-3-1新東京ビル4F

Web: https://www.qt.io/jp/
Email: japan@qt.io
TEL: 03-6264-4500