basysKom

QT UND DAS WEB

Eine Bestandsaufnahme

HMI-Entwicklerforum, Stuttgart 2019

Über uns

basysKom GmbH

- Qt Service Partner seit 2004
- Büros in Darmstadt und Nürnberg
- ~30 Mitarbeiter
- Teil der Münchner UX Gruppe
- Consulting, Training, Coaching und Entwicklungsdienstleitungen im Kontext Qt und Web-Technologien
- Fokusiert auf HMIs für Technische/Embedded-Anwendungen

Frank Meerkötter

- Development Lead @ basysKom

Einführung

In seinen Ursprüngen ist das Web mittlerweile 30 Jahre alt!

Hypertext, HTML, HTTP(S), Webbrowser, Webserver, (...) sind allgegenwärtig

Durch seinen riesigen Erfolg hat das Web einen starken Einfluss auf Software-Welt

Für eine ganze Reihe von Use-Cases ist der Einsatz von Web-Technologie der Default

Ist das Web der Heilige Gral?

Probleme beim Einsatz von Web-Technologien

- Stark Fragmentiertes Ökosystem (eigentlich mehrere Ökosysteme)
- Viele Änderungen (Stabilität von APIs/Frameworks, Modewellen)
- Offline muss immer nochmal gesondert betrachtet werden
- Auf schwacher Hardware nur begrenzt einsetzbar
  - Aufwendige System-Integration
  - Performance

It's a web's world

→ auch in 2019 gibt es gute, valide Gründe HMIs/Anwendungen mit nativen Frameworks zu entwickeln

Aber ein natives Framework muss sich ggfls. im Kontext einer durch das Web geprägten Welt beweisen

- Nutzung von Web-Diensten (Qt als Client)
- Anbieten von Web-Diensten (Qt als Server)
- Darstellung von Web-Inhalten mit Qt
- Qt im Browser

QT ALS CLIENT

Use-Cases: Qt als Client

Aufgabe

- Einbindung von Web-fokusierten Diensten

Beispiele

- Upload/download von Dateien/Daten
- Nutzung von REST- und/oder WebSocket-APIs

Zu beachten

- Serialisierung/Deserialisierung von API-Daten
- Transport-Sicherheit
- Authentifizierung/Autorisierung

HTTP(s)-Client

QNetworkAcccessManager

- C++ API
- Teil von QtNetwork
- Ein etwas abstrakter Name für einen HttpClient? ;-)
- Die API mag sich etwas altbacken anfühlen wenn man Coroutines/Promises oder ähnliches gewöhnt ist
  - In Qt seit 4.4

XmlHttpRequest

- JS-API die aus QML heraus genutzt werden kann
- weitgehend analog zur Browser-API
- Kann genutzt werden um aus einer Qt Quick Anwendung heraus Daten zu ziehen (oder mit einer API zu arbeiten), ohne C++ schreiben zu müssen

Beide unterstützen SSL/TLS (via OpenSSL)

WebSocket-Client

WebSockets == Bidirektionale Client/Server Verbindung

- Ermöglicht es dem Server Events in den Client (z.B. Webbrowser) zu „pushen“

WebSockets in Qt

- QWebSocket stellt einen Client zur Verfügung
- Teil eines eigenen QWebSockets Module
- C++ und QML API
- Unterstützt TLS (wss://)

Serialisierung/Deserialisierung

JSON

- das typische Datenformat für Web-APIs
- Unterstützt in Qt durch QJson
- Teil von QtCore
- keine Unterstützung für JSON-Schema (es gibt weitere Bibliotheken Dritter die mit QJson kombiniert werden können)

XML

- Old-school, aber immer noch sehr weit verbreitet

Optionen in Qt

- QXMLStream{Reader|Writer} in Qt Core
- QDomDocument in Qt XML
- Qt XML Patterns für einen validierenden Parser

Authentifizierung/Autorisierung

Clients müssen sich bei Web-Diensten authentifizieren

- Im Idealfall ohne in jedem Client Username/Passwort einzubinden
- Im Idealfall ohne einem Client Zugriff auf alle Daten in einem Account zu gewähren

Token-basierende Authentifizierung/Autorisierung

Unterstützung in Qt durch das QNetworkAuthorization Modul

- OAuth1/2
- Lizenz: GPL + Kommerziell

QT ALS SERVER

Use-Case: Qt als Server

Aufgabe

- Anbieten von REST und/oder WebSocket-APIs aus einer Qt-Anwendung heraus

Beispiele

- Schnittstellen für Test-Automatisierung
- Headless Qt (Qt als „freundliches“ C++ auf einem Embedded-System)
- Integrationsschnittstelle mit IT-Systemen (z.B. Import von Aufträgen)
- Integration von Legacy-Anwendungen (C++/Qt) in eine Web-Umgebung

Use-Case: Qt als Server

Lösung: QHttpServer!

- eine lange vermisste Komponente
- es gibt eine ganze Reihe von Qt-Style HttpServer Bibliotheken auf github – leider keine mit einer kritischen Masse an Nutzern/Entwicklern

Lösung: QWebSocketServer

- Teil des Qt WebSocket Moduls
- Verfügbar seit Qt 5.4
- C++ und QML bindings

Jetzt als „offizieller“ Teil von Qt

- https://code.qt.io/cgit/qt-labs/qthttpserver.git/
- Leider “nur” ein Labs-Projekt
- Lizenz: GPL + Kommerziell

DARSTELLUNG VON WEBINHALTEN MIT QT

Use-case: Einbindung von Web-Inhalten

Aufgabe

- Existierende Web-Inhalte sollen in eine Qt-Anwendung eingebunden werden
- Beispiele
  - Generischer Webbrowser
  - Walled-Gardens
  - Spiele
  - Reportings
  - Online-Shops
  - ...

Lösung

- QWebEngine
  - Einbindbare Browser-Komponente basierend auf Chromium
  - QML & C++ APIs
  - Relativ komplexe API – ermöglicht komplexe Anwendungen
- QWebView
  - Wrapper für die nativen Browser-Komponenten auf Android/iOS
  - Nutzt QWebEngine auf anderen Plattformen als Fallback
  - QML only
  - Recht einfache API – erlaubt eine High-level Integration

Use-case: Web-HMI & Qt/C++ backend

Aufgabe

- Mixed-Mode-Anwendung. Frontend mit Web-Technologien, Backend in Qt/C++
- Abgrenzung zu einem klassischen Client/Server Szenario: tiefere Integration

- Motivation
  - Performance
  - Legacy-Code (aka. „valuable business logic“)

Lösung

- QWebEngine als Laufzeit für das HMI
- QWebChannel um aus Browser-JS auf C++ QObjects im Backend zugreifen zu können
- Siehe: doc.qt.io/qt-5/qwebchannel.html

Use-case: Webrowser für Embedded-HMIs

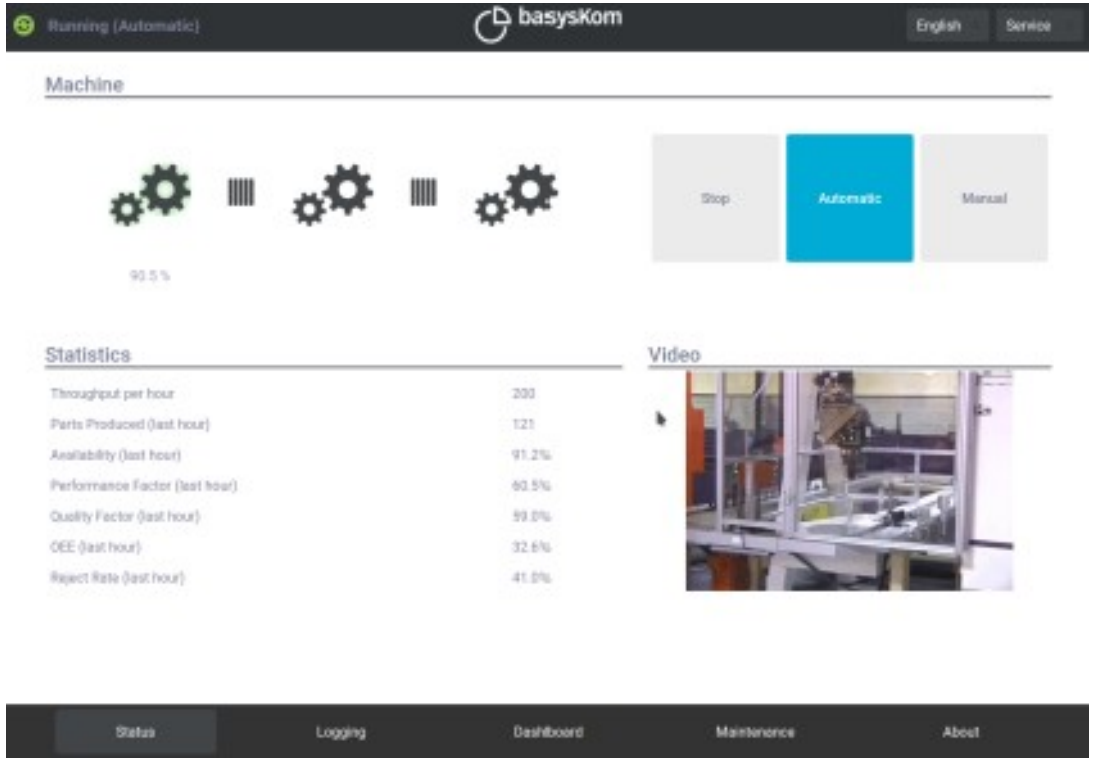
Aufgabe

- Embedded-HMI
- Full-screen, kein Window-Manager
- Eigentliche keine “Qt-Anwendung” ;-)

Lösung: Qt WebEngine (wiedereinmal)

Fallstricke

- Systemintegration
- Große Komponente mit vielen Abhängigkeiten
- Benötigt oft stärkere Hardware (im vgl. zu Qt Quick)

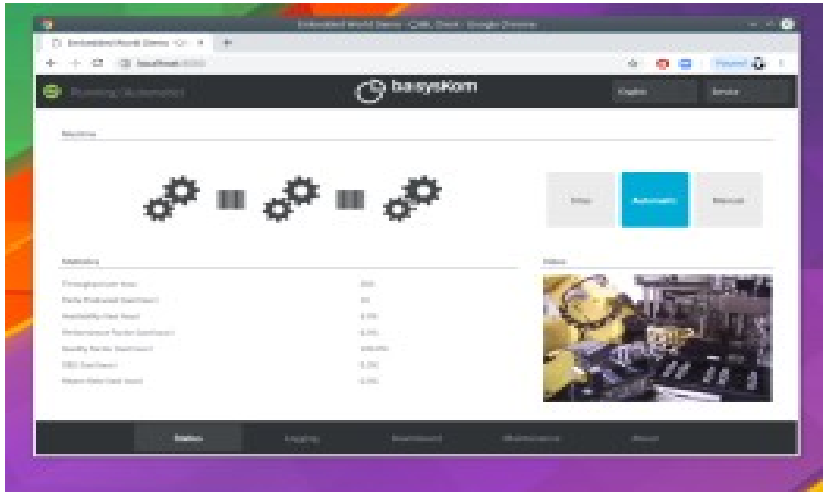


QT IM BROWSER

Use-case: Browser als Remote-Display

Aufgabe

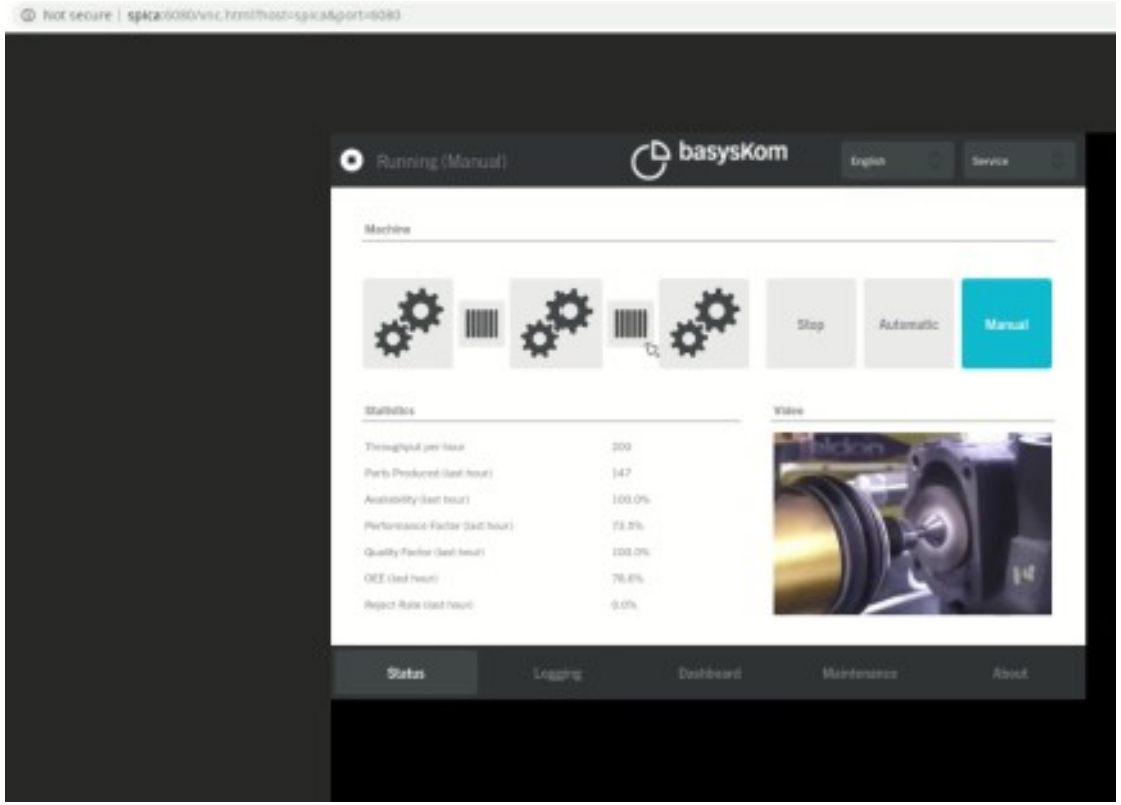
- Darstellung eines Qt-HMI im Webbrowser
  - Geräte ohne Display
  - Remote-Zugriff auf ein Gerät mit Display



Lösung #1: WebGL streaming

- Qt QPA das WebGL-Befehle per WebSocket überträgt
  - myApp -platform webgl
- Nur Qt Quick, kein QWidget
- Lizenz: GPL + Kommerziell
- Auflösungsunabhängig
- Latenzanfällig
- Jeweils nur ein Nutzer gleichzeitig
- Beispiel: https://www.qt.io/bosch-built-with-qt

Use-case: Browser als Remote-Display (Trick 17)



Lösung #2: VNC

- VNC QPA + Websocket-Proxy + NoVNC (JS-Client)
- QWidget & Qt Quick (SW-renderer)
- Überträgt Bitmaps
- Auflösung ist fix
- Anfällig für Latenz
- Auch hier – immer nur ein Nutzer gleichzeit

Use-case: Qt im Browser

Aufgabe

- Eine Qt-Anwendung soll in einem Standard-Webbrowser ausgeführt werden
- Es soll nichts zusätzlich installiert werden müssen

Beispiele

- Geräte ohne Display (die Ihre Service-Anwendung selbst ausliefern)
- Verteilung von Inhouse-Software

Exkurs: WebAssembly

- Portable Laufzeitumgebung (Byte-Code, APIs)
- In allen aktuellen Browsern verfügbar
- Wichtig: eine WebAssembly-Anwendung unterliegt den selben Sicherheitsregeln wie eine Webseite

Use-case: Qt im Browser

Lösung: Qt for WebAssembly

- Offiziell unterstützt mit Qt 5.13
- www.qt.io/qt-examples-for-webassembly

Relativ große Downloads (~25MB für eine einfache Qt Quick Anwendung)

Linker ist sehr langsam

Anwendungen unterliegen den Einschränkungen der Browser-Sandbox

- kein Zugriff auf das Dateisystem
- native APIs
- Netzwerkzugriff ist auf HTTP und WebSocket beschränkt

ZUSAMMENFASSUNG

Zusammenfassung

Vier Perspektiven auf “Qt und das Web”

- Client
- Server
- Darstellung von Web-Inhalten in Qt
- Qt im Webbrowser

Weitere Aspekte

- ADL, Swagger, openAPI
- Templating, HTML Generierung
- Webanwendungen in Qt?
- …

Berichten Sie von Ihrem „Qt and das Web“ Projekt

- Besuchen Sie uns am basysKom-Stand

FRAGEN?

Frank Meerkötter
Development Lead
frank.meerkoetter@basyskom.com