Accessing Middleware from Qt, QML or HTML5

basysKom

basysKom... introducing ourselves

Embedded Software Engineering

– Partner for the development of innovative quality products

Expertise in...

– Embedded R&D

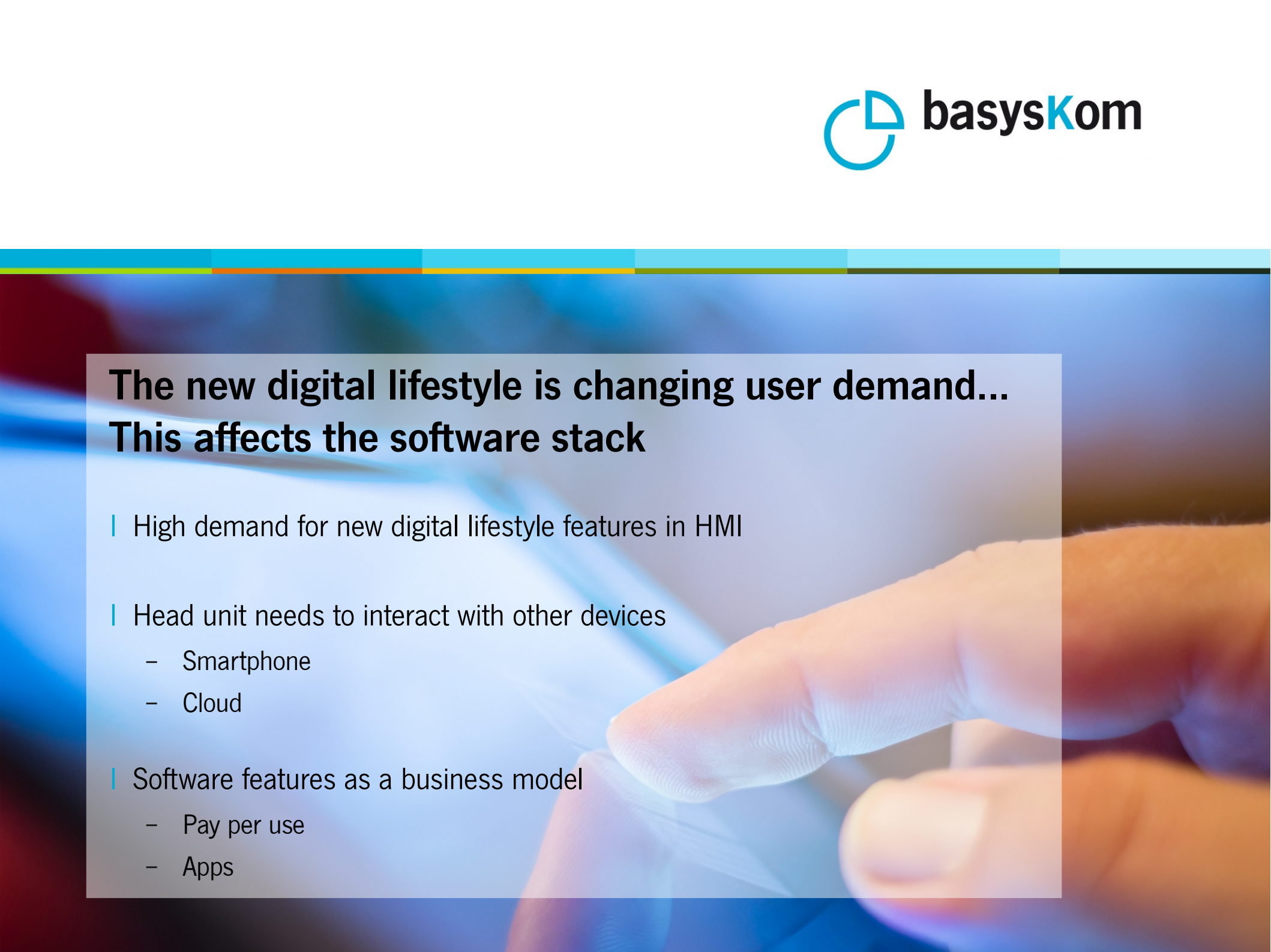
– Embedded architecture design, middleware, HMI

– Open source standards & Linux stack

Located in Germany, Darmstadt + Nürnberg

Our Offering:

– consulting	on strategic and operational selection and architecture of embedded technology and development processes
– solution delivery	R&D including delivery of full components or services requiring specialist knowhow



The new digital lifestyle is changing user demand... This affects the software stack

- High demand for new digital lifestyle features in HMI
- Head unit needs to interact with other devices
  - Smartphone
  - Cloud
- Software features as a business model
  - Pay per use
  - Apps

New Requirements to Software Architecture

1 – Reduce time to market

2 – Increase flexibility

- OS independency
- Device independency
- UI independency

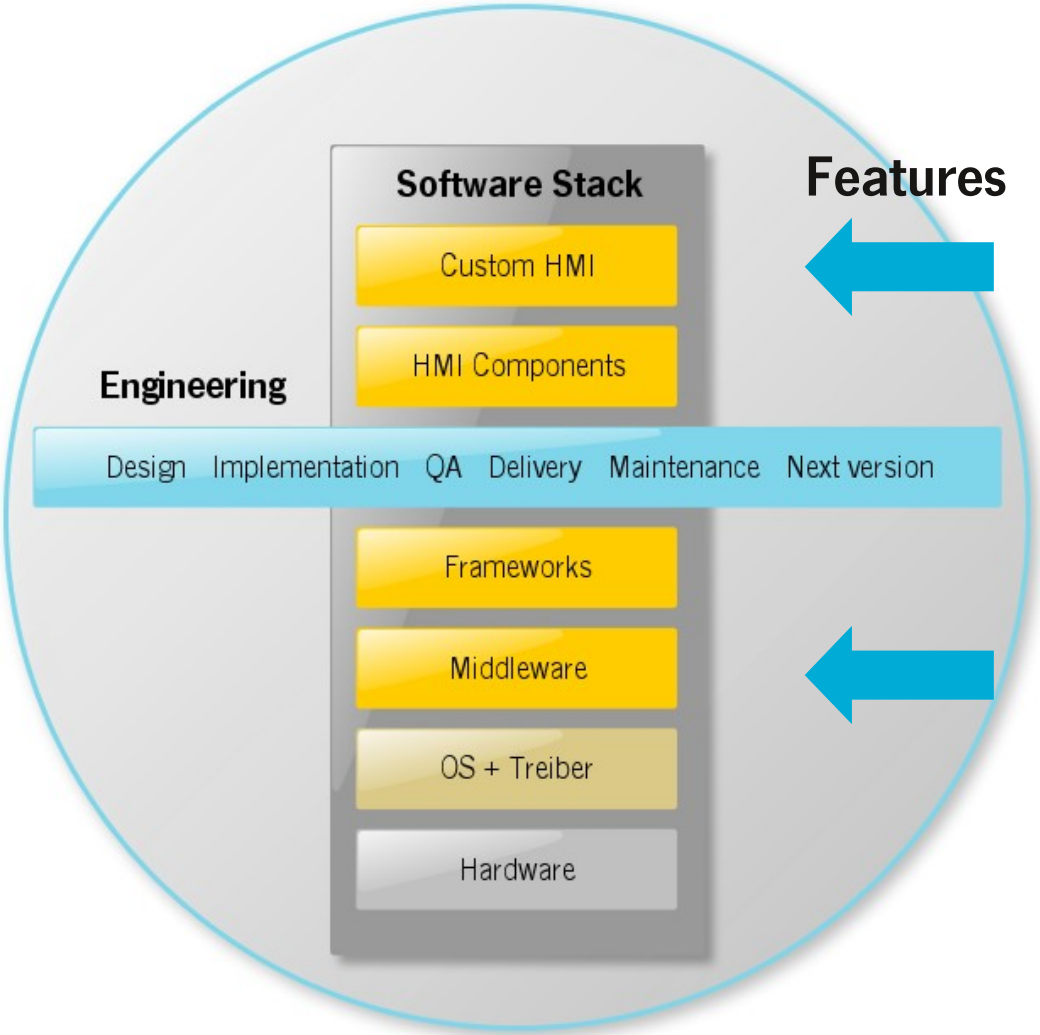
3 – Improve quality of software

- Usability
- Security
- Stability
- Scalability

Sustainable Software Architecture

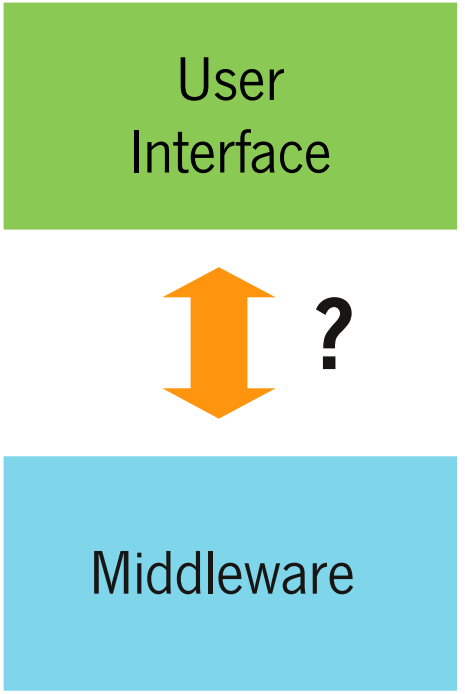
Solution: Reuse of Software

- **Human Machine Interface**
  - Custom User Functionality
  - Application Logic
  - Design
- **Middleware / Backends**
  - Access to Custom System Functionality
- **Standard Components**
  - UI Framework, backends, OS, drivers, etc.
- **Middleware is the Glue!**



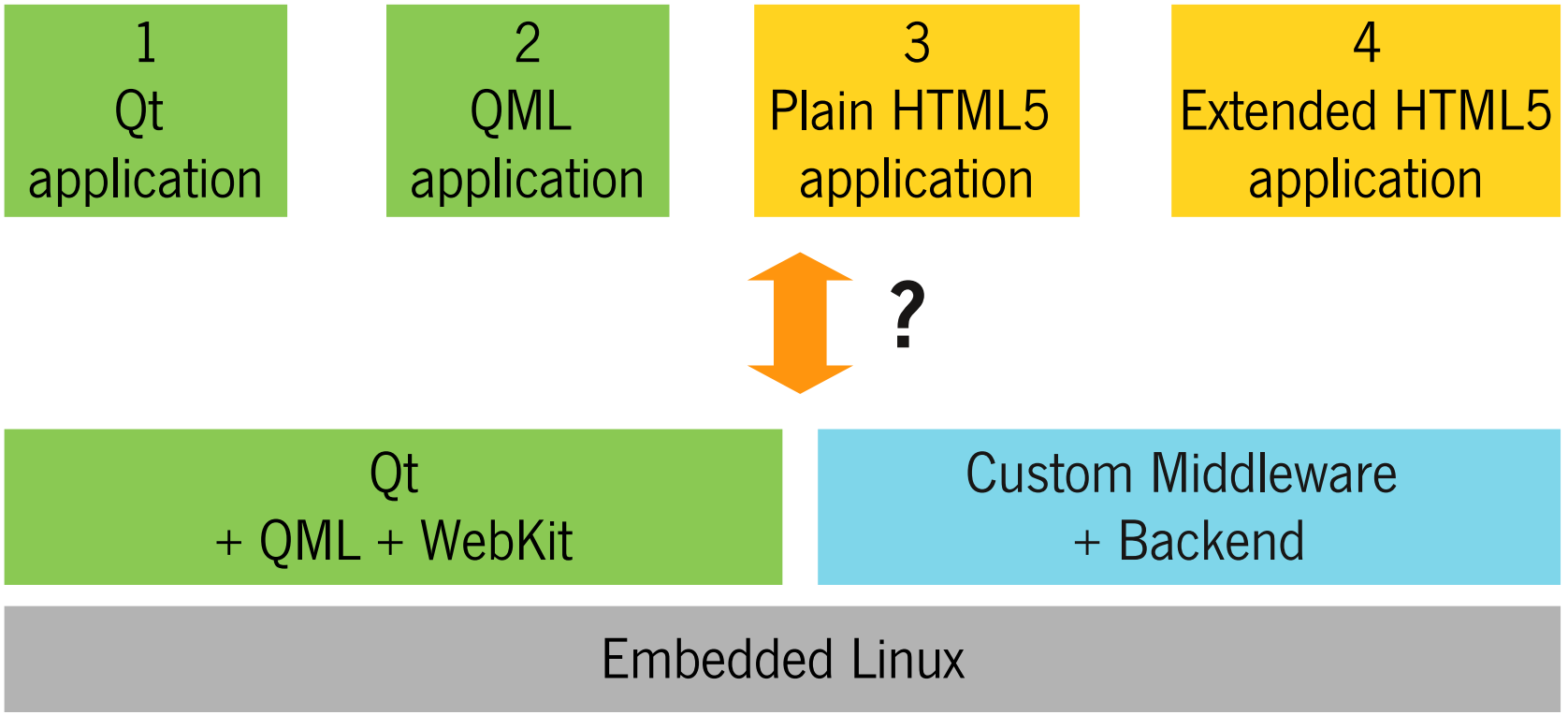
Requirements to Middleware

- Technology Choice Middleware
  - C/C++
  - Via IPC
- How to access Middleware?
  - Little coding overhead
  - Speed
  - Robust, Type safety
  - Reusable, maintainable



Alternative Technologies for Embedded HMI

- Access to middleware depends on HMI technology
- Alternatives:



basysKom

Middleware Access from Qt & QML

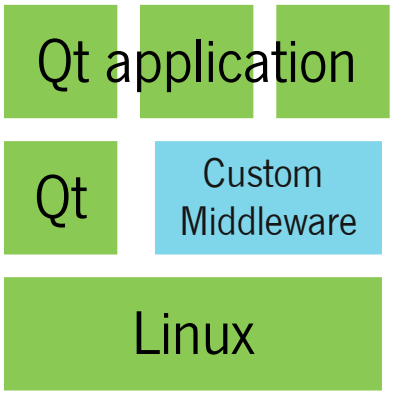
Middleware Access from Qt UI

About Qt

- C++ Development Framework
- Many libraries and modules

C++ => Straightforward Middleware Integration

- Through function calls
- Signals & slots
- IPC, e.g. DBUS



Example - Middleware

- Small Library
- Inherits from QObject
- Triggers actions
- Sets values, notifies about changes
- Abstracts backend

```
1   #include "middleware.h"
2
3   #include <QDebug>
4
5   MiddlewareObject::MiddlewareObject(QObject *parent) :
6       QObject(parent)
7   {
8   }
9
10  void MiddlewareObject::activate() {
11      qDebug() << "activated";
12      /*
13       * activateLaser();
14       *
15       */
16  }
17
```

Example – Qt UI

- Uses Our Middleware Library
- Creates GUI
- Access Middleware

```
1  #include <QApplication>
2  #include <QPushButton>
3
4  #include "qt_ui.h"
5  #include "middleware.h"
6
7  int main(int argc, char *argv[])
8  {
9      QApplication app(argc, argv);
10
11     QPushButton activate("Activate");
12     activate.resize(100, 30);
13
14     MiddlewareObject mw;
15     QObject::connect(&activate, SIGNAL(clicked()),
16                      &mw, SLOT(activate()));
17
18     activate.show();
19     return app.exec();
20 }
21
```

Evaluation Qt

- High-quality, stable and well-tested framework APIs
  - Touchscreen/animation complicated
  - More suited for conventional desktop applications

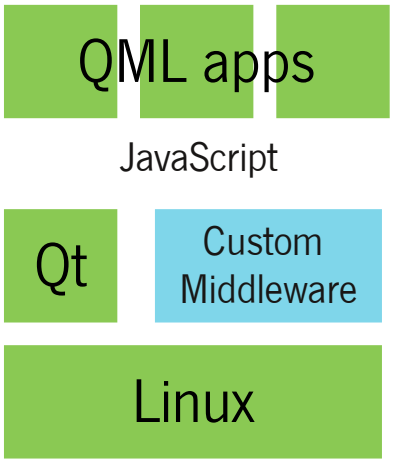
- Pros
  - Very flexible connectivity
  - Frontend and backend can be managed in the same programming language
  - One-stop shop

- Cons
  - Cumbersome touch support
  - Imperative UI 'design' (or use Designer)
  - Only 'conventional' UI paradigms implemented
  - Classic UI design, not very design centric

Middleware Access from QML

About QML

- Design centric
- Simple And Fast
- Low Barrier To Entry
- Animations come at no cost
- Next generation QML uses modern GPU acceleration
- As QML is part of Qt it can be extended with classic Qt technologies (C++, QMetaObject system)



Example – QML UI

- On the C++ side:
  - Use Qt's MetaObject system to configure middleware
  - Properties to set and read values
  - Final properties to read states

```
class Machine : public QObject
{
    Q_OBJECT

    // configure the machine
    Q_PROPERTY(int rotationalSpeed READ rotationalSpeed WRITE setRotationalSpeed NOTIFY rotationalSpeedChanged)
    // get read only values
    Q_PROPERTY(QString state READ state NOTIFY stateChanged FINAL)
    Q_PROPERTY(QStringList someList READ someList NOTIFY someListChanged FINAL)
```

- Expose actions with Q_INVOKABLE

```
Q_INVOKABLE void activate();
```

Example – QML UI

- Don't forget to emit signals whenever your underlying model changes:
  - Get property bindings for free

```
void Machine::setRotationalSpeed(int mRotationalSpeed)
{
    if (m_rotationalSpeed != mRotationalSpeed) {
        m_rotationalSpeed = mRotationalSpeed;
        // access middleware
        // do something to bring the motor up to speed ...
        // ...

        // notify QML to guarantee property binding
        emit rotationalSpeedChanged();
    }
}
```

Example – QML UI

- Expose middleware as a plugin
- Use qmlRegisterType<Type>() to make your api visible in QML
- Use QmlDir to export module

```
class MiddlewarePlugin : public QQmlExtensionPlugin
{

    Q_OBJECT
    Q_PLUGIN_METADATA(IID "org.qt-project.Qt.QQmlExtensionInterface")

public:
     MiddlewarePlugin(QObject *parent = 0)
         : QQmlExtensionPlugin(parent)
     {

     }

     void registerTypes(const char *uri)
     {
         Q_ASSERT(QLatin1String(uri) == QLatin1String("Middleware"));
         qmlRegisterType<Machine>(uri, 1, 0, "Machine");
     }

};

#include "middlewareplugin.moc"
```

```
1 module middleware
2 plugin middlewareplugin
3
```

Example – QML UI

- Example QML file
- Import the module
- The formerly designed classes act as an interface to the middleware

```
import QtQuick 2.0
import middleware 1.0

Rectangle {
    id: root
    width: 800; height: 480

    Text {
        anchors.centerIn: parent
        text: myMachine.state
    }

    Button {
        text: "Click me"
        anchors.bottom: parent.bottom
        anchors.horizontalCenter: parent.horizontalCenter
        onClicked: myMachine.activate()
    }

    Machine {
        id: myMachine
        rotationalSpeed: 2400
    }

}
```

Models

- Easy integration of Qt's data structures and models (QList, QAbstractItemModel)
- Example: Declare a Q_PROPERTY as part of your QML object:

```
Q_PROPERTY(QStringList someList READ someList NOTIFY someListChanged FINAL)
```

- Use it in QML:

```
ListView {
    anchors.centerIn:  parent
    width: 250; height: 300
    model: myMachine.someList
    delegate: Component {
        Rectangle {
            width: parent.width;
            height: 60
            color: "#000"
            anchors.margins:  5
            Text {
                anchors.centerIn:  parent
                font.pointSize:  12
                text: modelData
                color: "#822"
            }
        }
    }
}
```

Models

- The other way around:
  - Export your models as context properties
  - attach them to your global namespace

```
// get global parameters (for example an QAbstractItemModel) from middleware
// and set it as a context property
if(qmlView.rootContext()) {
    qmlView.rootContext()->setContextProperty("globalParameters", middlewareParameters);
}
```

  - Use it in QML

```
ListView {
    anchors.centerIn:  parent
    width: 250; height: 300
    model: globalParameters
    delegate: Component {

    ...
    ...
```

Evaluation QML

Pros

- Uses Qt's meta object system
- Access to all Qt + middleware functions
- Easy integration via signal and slots
- Properly designed middleware can be shared between Qt and QML frontends

Cons

- More languages means a more complex technology stack (QML, JavaScript, C++)
- Higher minimum requirements (OpenGL ES 2.0)

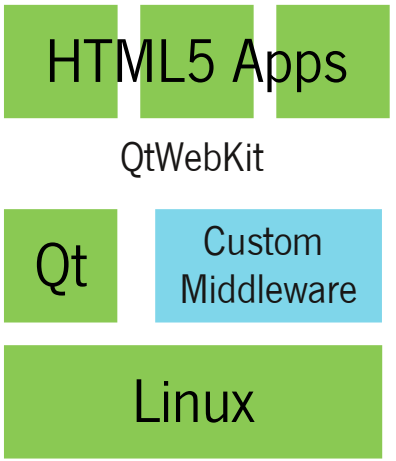
basysKom

Middleware Access from HTML5

Middleware Access from HTML5 – An Overview

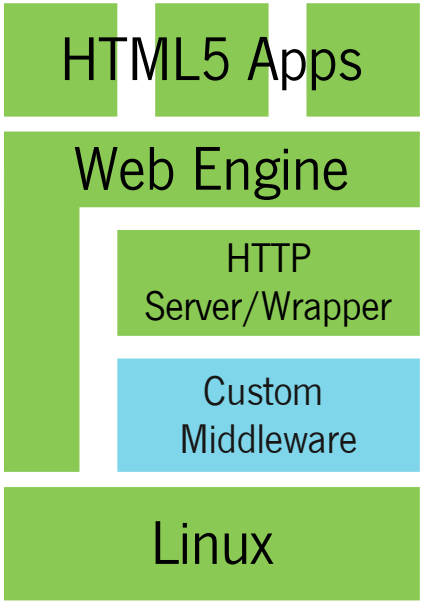
About HTML5

- QtWebKit is an essential module of Qt5
- Development in HTML/CSS/JavaScript
- Very large HTML5 designer/developer community
- Modern UI features (e.g. multitouch/accelerated animation)
- HTML5 apps are usable on standard devices like tablets/mobiles/etc.



Plain HTML5 – Middleware Access

- **Today: XMLHttpRequest**
  - High level/high latency
  - No persistent connections
- **Tomorrow: Websockets**
  - Fix XMLHttpRequest shortcomings
  - Might come to QML (QTBUG-26298)
- **How To Use A Custom Embedded Middleware**
  - Middleware uses HTTP server (or wrapper)



Plain HTML Example - XMLHttpRequest

- Daily bread of a web developer

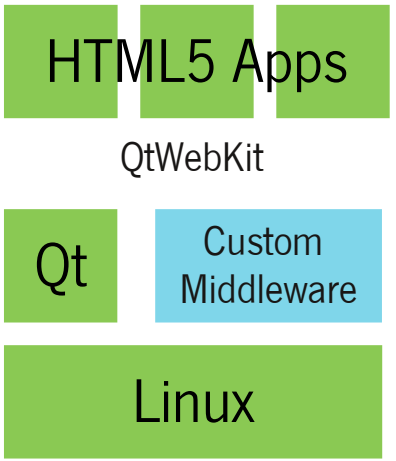
```
1  xmlhttp = new XMLHttpRequest();
2  xmlhttp.onreadystatechange = function() {
3      if (xmlhttp.readyState==4 && xmlhttp.status==200) {
4          alert(xmlhttp.responseText);
5      }
6  }
7  xmlhttp.open("GET", url, true);
8  xmlhttp.send();
9
```

- Noone uses XMLHttpRequest
- Everyone uses libraries (jQuery/YUI/...)

Extended HTML5 – Middleware Access

Extended Runtime using QtWebKit

- Use full Qt capabilities to connect middleware
- Provide APIs to JavaScript context
- Directly interact with Apps (Call JavaScript code from Qt)
- Connect signals/slots in your JavaScript code



Extended HTML Example – Triggering A Qt Function

- Middleware C++ library
- Own Qt app with QWebView
- HTML File

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>
5              Qt Middleware Demo
6          </title>
7      </head>
8      <body>
9          <button onclick="middleware.activate()">Click me.</button>
10     </body>
11 </html>
12
```

- Add middleware object to JS context
- Call Q_INVOKABLEs

```
MiddlewareObject mw;
QWebView webView;

webView.load(QUrl("main.html"));
webView.page()->mainFrame()->addToJavaScriptWindowObject("middleware", &mw);
```

Extended HTML Example 2 – QStringListModel + Knockout.js

- **What about more complex data?**
  - Qt models are not directly usable
- **Example with a QStringListModel**
- **Use Knockout.js (http://knockoutjs.com/)**
  - Model-View-View-Model JavaScript library
- **Functionality**
  - Changes on a web-page get immediately propagated to the C++ backend
  - This could be used to e.g. directly edit system settings

Extended HTML Example 2 – QstringListModel + Knockout.js

- Web runtime exposes middleware stub into JS context

```
1131        m_webView->page()->mainFrame()->addToJavaScriptWindowObject("Qt", this);
1132        m_webView->page()->mainFrame()->addToJavaScriptWindowObject("console", m_stub);
1133        m_webView->page()->mainFrame()->addToJavaScriptWindowObject("middleware", m_stub);
```

- Stub provides access to model and update function
- Data is automatically converted to/from JavaScript

```
19  QStringList MiddleWareStub::model()
20  {
21      m_initialized = true;
22      return m_model->stringList();
23  }
```

```
30  void MiddleWareStub::update(QStringList data)
31  {
32      if(m_initialized) {
33          m_model->setStringList(data);
34      }
35  }
```

Extended HTML Example 2 – QstringListModel + Knockout.js

- The HTML side: Initialize Knockout with model

```
51          self.reloadModel = function() {
52              if(typeof(window.middleware) != 'undefined') {
53                  middleware.reload();
54                  self.entries(middleware.model());
55              }
56          }
```

- Whenever entries are added/removed, Knockout tells the stub

```
60          ko.computed(function() {
61              console.log('updating backend');
62              if(typeof(window.middleware) != 'undefined') {
63                  middleware.update(ko.toJS(self.entries));
64              }
65          });
```

- Reverse direction can be implemented by connecting signals in JavaScript (e.g. dataChanged)

Evaluation HTML5 On QtWebKit

- It is possible to write interesting HMI using HTML5

- **Pros**
  - Existing HTML5 apps will run out-of-the-box
  - Access to Qt/middleware functions is possible (runtime modification)

- **Cons**
  - Lots of mixed technologies (HTML/CSS/JavaScript/C++)
  - Direct access to Qt/middleware functions requires runtime modification
    - Leads to non-portable apps
  - Translation layer for more complex models required
    - No direct usage of Qt models

basysKom

Conclusion

- **Qt**
  straight forward C++ integration of middleware
  not suitable for touch UI
- **QML**
  very quick creation of custom HMI
  middleware integration through Qt mechanisms
- **HTML5**
  broad developer base
  more complicated to connect to middleware

basysKom

Conclusion

- **Big advantage of Qt: Hybrid use**
  - Native Qt
  - QML
  - HTML5

- **Clear Middleware – HMI separation enables**
  - software reuse
  - Quick HMI creation
  - Fast time-to-market of new use cases

Dr. Eva Brucherseifer
Managing Director & Sales
eva.brucherseifer@basyskom.com

Berthold Krevert
Software Engineer
berthold.krevert@basyskom.com

Sumedha Widyadharma
Senior Embedded Engineer
sumedha.widyadharma@basyskom.com

basysKom GmbH
Robert-Bosch-Str. 7
64293 Darmstadt
Germany

sales@basyskom.com
+49 (6151) 870 589 101

www.basyskom.com