Device Tailored Compositors

with the QtWayland-Compositor Framework

Andreas Cord-Landwehr

February 5, 2017

FOSDEM, Brussels

Introduction

About Me & the Talk

About Me

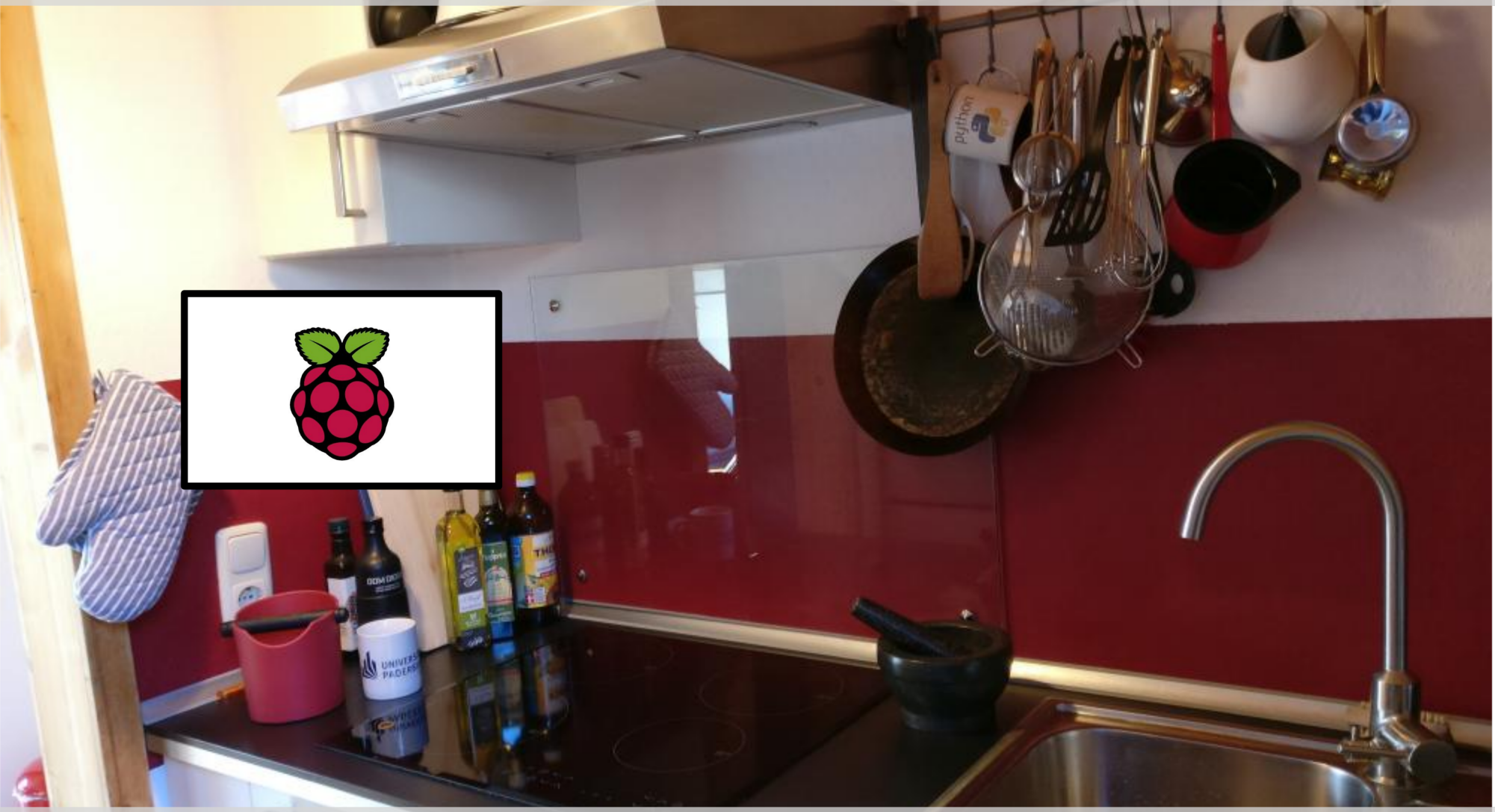
- IRC-nick: CoLa
- KDE developer since ≈ 2010
- did PhD in algorithmic game theory at Paderborn University
- now working as developer at CLAAS E-Systems in department for displays and operator panels for big agriculture machines

This talk is about the QtWayland Compositor Framework:

1. topic is between my KDE and my professional work
2. practical introduction into how to create your own compositor
3. the talk shall make you eager to experiment with the QtWaylandCompositor
4. want to convince you that there is a new solution for many embedded device needs

The Red Thread

Let's say, I need a terminal for helping me in the kitchen...



Setting & Requirements

- several applications shall run on the device
  - cooking eggs timer app
  - tea timer app
  - current time app
- seemless UI between the application windows
- swipe animation for switching applications
- want to use a standard embedded Linux based device with 3D acceleration (e.g. Raspberry Pi)

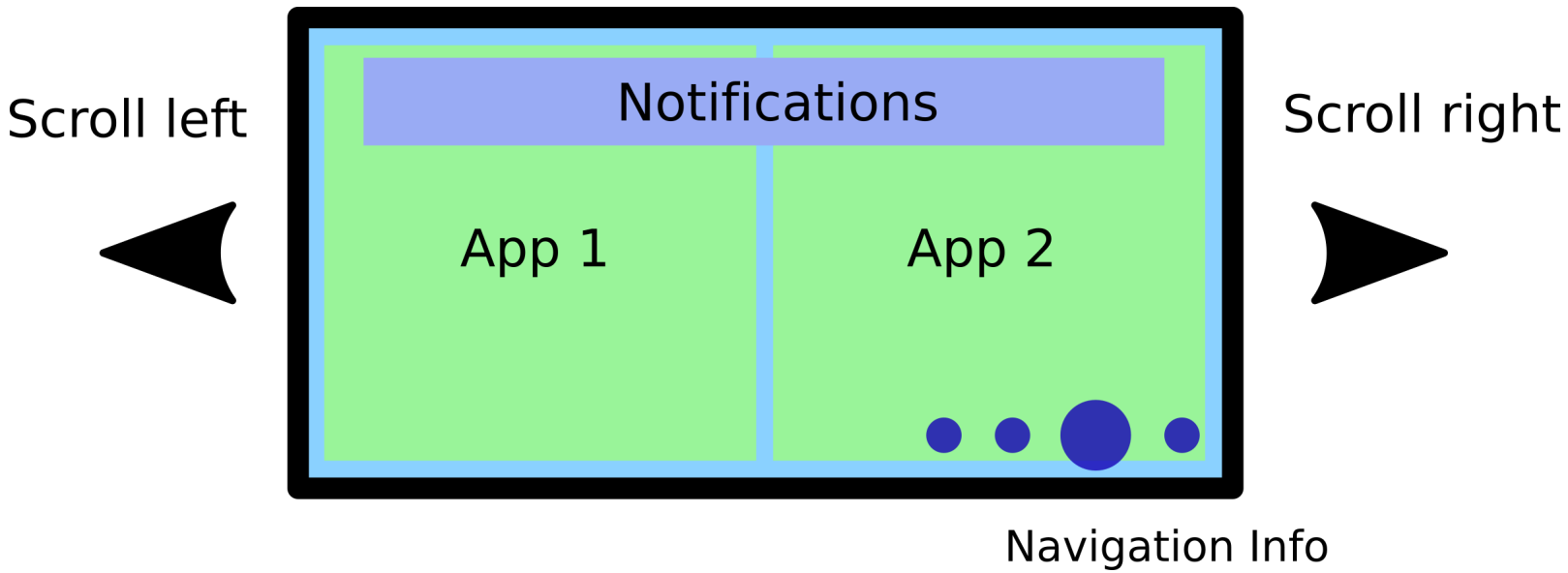
→ we need a (Wayland) compositor

→ we can use QtQuick

Yes, the above can be achieved in a much simpler way, but by exchanging these trivial apps with eg. internet radio, navigation system etc. and you get exactly what modern cars put onto their devices today.

Interaction Concept for the Kitchen Device

Something your designer might come up with



The remainder of the talk: how QtWayland helps to build this

What is Wayland, again?

A much too short answer for this question.

Wayland is a protocol specifying the communication between a compositor (display server) and its clients (applications with their windows)

- in the embedded world, Wayland is the already established successor of X
- there are several Wayland compositor implementations:
  - Weston: the reference implementation
  - for desktop environments: KWin, Mutter, Enlightment ...
  - some propriatary compositors by device vendors exist
- protocol extensions add further functionality to the Wayland protocol:
  - specified in XML files, code generated via wayland-scanner
- available shells:

wl-shell	default protocol for window handling, already introduced with Wayland 1.0
xdg-shell	successor of wl_shell with implementations provided by individual compositors
ivi-shell	protocol specifically for special automative form factors (IVI = in-vehicle infotainment), used via the ivi-controller protocol extension

The Qt Wayland Compositer API

http://doc.qt.io/qt-5/qtwaylandcompositor-index.html

- possible to write a compositor in just QML
  → QML is declarative language especially used for UI development on embedded devices/smartphones/touch applications
- supports Qt/C++ wrapper generation for Wayland protocol extensions
- supports multiple screen outputs
- provides wl-shell, xdg-shell and ivi-shell protocols
- History:
  - since many years, there was an internal (but cumbersome to use) API
  - Compositor API rewritten for Qt 5.7 (tech preview)
  - Stable API since Qt 5.8

Alternatives: What about using the IVI-Shell extension in kitchen scenario?

- protocol only suited for very static settings
- touch gestures and window animations hard to implement

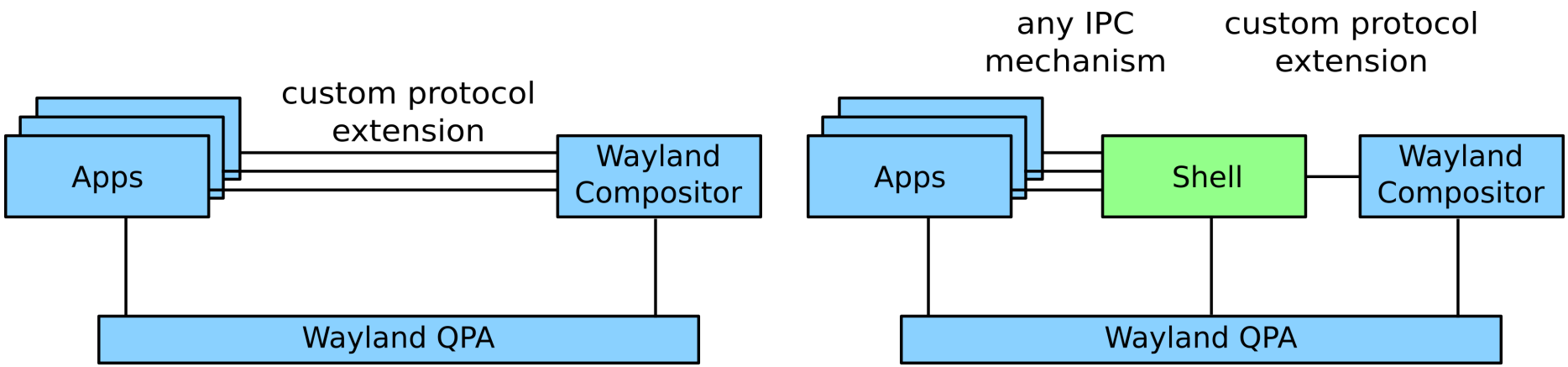
Possible System Architectures for my Kitchen Device

Components:

apps Qt application running as Wayland client (with Qt: running on Wayland QPA with `-platform wayland`)

compositor application acting as Wayland Server

shell (optional, but IMO very reasonable) application that handles the more complex compositing logic and encapsulates all Wayland communication; eg. filters and sorts all safety relvant notifications from your applications



Note: in this talk I will stick to the left variant

One-Slide Wayland Compositor

```
import QtQuick 2.0
import QtQuick.Window 2.2
import QtWayland.Compositor 1.0
import Fosdemdemo2017 1.0

WaylandCompositor {
    id: demoCompositor
    WaylandOutput {
        compositor: demoCompositor
        sizeFollowsWindow: true
        window: Window {
            width: 800; height: 400; visible: true
            Rectangle {
                anchors.fill: parent
                ListView {
                    anchors.fill: parent
                    model: ListModel { id: listModel }
                    orientation: ListView.Horizontal
                    delegate: ShellSurfaceItem {
                        shellSurface: model.shellSurface
                        onSurfaceDestroyed: { listModel.remove(index) }
                    }
                }
            }
        }
    }
    WlShell {
        onWlShellSurfaceCreated: {
            listModel.append({"shellSurface": shellSurface});
} } }
```

Essential Components (1/2)

The Compositor and the Shell

```
WaylandCompositor {
    WaylandOutput { ... }
    WlShell {
        onWlShellSurfaceCreated: {
            // handle shellSurface object
            listModel.append({"shellSurface": shellSurface});
        }
    }
}
```

WaylandCompositor

- representation of the compositor
- usually the root object of the scene
- always should have output and shell extension

Shell Extension

- the protocol interface that gives access to the surfaces
- WlShell and XdgShell are supported
- handle the onWlSurfaceCreated() signal

Essential Components (2/2)

The Surface Items and the Output

```
WaylandOutput {
    compositor: demoCompositor
    sizeFollowsWindow: true
    window: Window {
        width: 800; height: 400; visible: true
        ListView {
            anchors.fill: parent
            model: ListModel { id: listModel }
            orientation: ListView.Horizontal
            delegate: ShellSurfaceItem {
                shellSurface: model.shellSurface
                onSurfaceDestroyed: { listModel.remove(index) }
}   }   }   }
```

ShellSurfaceItem and QWaylandQuickItem

- wrapper around shell surfaces to handle them like ordinary QtQuick Items
- visibility and input behavior follows typical QtQuick mechanisms

Output

- manages rectangular output region in which content can be shown
- compositor can have multiple outputs

Protocol Extension for Alarm Notifications (1/2)

How to add your own custom Wayland protocol?

Code: https://github.com/cordlandwehr/fosdem-2017-talk-qtwayland/tree/master/demo

The Custom Protocol Extension

```
<protocol>
    <interface name="demo_extension" version="1">

        <request name="notification">
          <description summary="Example Notification Event">
            Example request from client to server
          </description>

          <arg name="text" type="string"/>
        </request>
    </interface>
</protocol>
```

Protocol Extension for Alarm Notifications (2/2)

How to add your own custom Wayland protocol?

On the Compositor Side

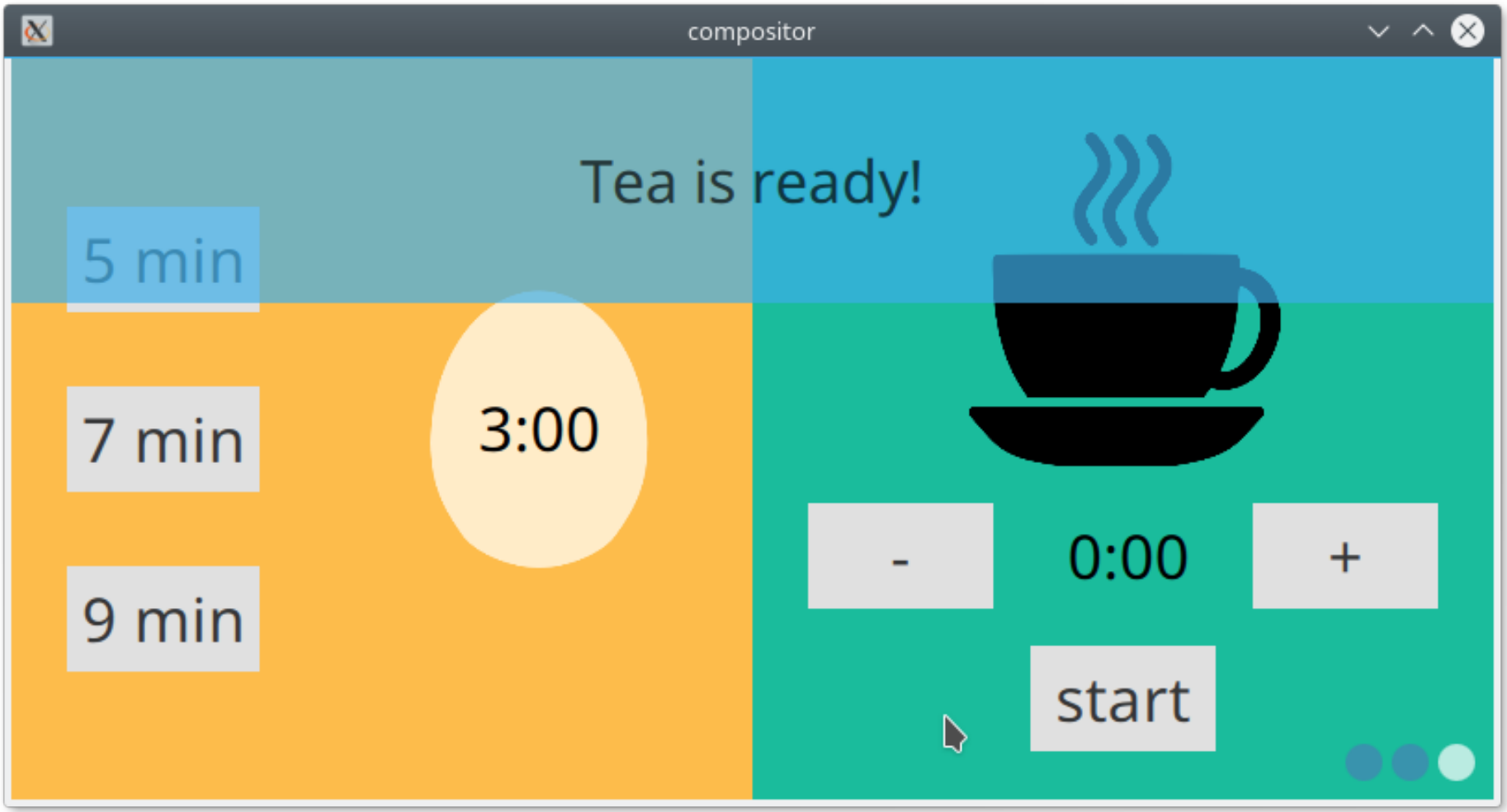
1. create QWaylandCompositorExtensionTemplate derived protocol wrapper
2. use qtwayland-scanner to generate Qt & C++ binding classes for protocol
3. add CustomExtension in the WaylandCompositor element in the QtQuick context and connect to signals/use methods for sending data

On the Client (Application) Side

1. create QWaylandClientExtensionTemplate derived protocol wrapper
2. use qtwayland-scanner to generate Qt & C++ binding classes for protocol
3. create client protocol wrapper instance and use it

Demo

Everything put together with some QtQuick UI sugar.



Conclusion

- the QtWayland Compositor framework drastically simplifies the creation of a compositor for a specific/special UX requirement
- prototyping such a compositor is just a matter of a day
- window compositing becomes UI design:
  - when a surface looks and behaves like a QtQuick item, you can handle it like a QtQuick item...
  - no special Wayland developer needed but "just" a QtQuick UI developer can develop your compositor tailored for your individual form factor

→ try it out!

Demo and all sources of this talk are available here:

`https://github.com/cordlandwehr/fosdem-2017-talk-qtwayland/tree/master/demo`

References

- Johan's "The Qt Wayland Compositor API" introduction talk at QtCon 2016
  https://conf.qtcon.org/en/qtcon/public/events/392
- Online Help
  http://doc.qt.io/qt-5/qtwaylandcompositor-index.html
- IRC at freenode: #qt-lighthouse

Thank you for your attention!

Andreas Cord-Landwehr
E-mail: cordlandwehr@kde.org