Code less.
Create more.
Deploy everywher

Qt Quick

Scalability in a dynamic world

How to handle multiple resolutions and form-factors

NOKIA

Nokia Device Portfolio and Software Evolution

- Nokia's current and future Device portfolio is streamlining
  - N900: 800 x 480
  - N97/5800/N8: 360 x 640
  - Other: HD-ready
- But also multiplying into new domains via MeeGo.com
  - Tablet, TV, Car
- Wildly varying Software Stacks have been built to accommodate Form Factors of the Handset device domain
  - Differing implementation, same problem
  - i18n, Operator Customization, DPI, Orientation and Keyboard policy
- Before the touch era, UI's were simpler to scale
  - Input mandated directional access – causing 'block' UI design
  - Comparable to a cell grid (e.g. a HTML table)
- Fluid and Differentiated UI's represents a new challenge

NOKIA

One Size Fits All

- Scalability assumes that UI's can be written to work everywhere

THIS IS NOT POSSIBLE

within reasonable limits

- Because
  - It imposes too much strain on the underlying framework to magically make the UI work in a broad set of environments
  - It will render the framework unusable to the developer, because it's impossible to know how the UI will actually behave
  - It depends on factors that are not controllable by the framework, such as availability of correctly sized graphic assets (SVG is not an option in all cases)
  - It eventually leads to bad compromises (such as pixelated, stretched images, clipped text, illegible graphics)

NOKIA

Many sizes fit most

- What's the alternative? Rewrite everything?

	Pros	Cons
Reuse	Write once Easier to maintain	More complex QA Sub-optimal results
Rewrite	Full flexibility to handle different use cases	More code to create, maintain and debug

- Different device domains require different UI paradigms
  - Viewing distance
  - Navigation model
  - Interaction model
- However, UI elements can to some extent be reused

NOKIA

Reuse and Rewrite!

- Application architecture allows reuse of common UI elements and refactoring of domain-specific paradigms
- Write reusable code to access data and handle App logic
  - This gives consistency and makes QA easier
  - This is aligned with QML's world view where Presentation / View and Business logic are separate, and business critical code resides in C++
- Write UI elements that can be reused
  - "Can I see others using my element?" rewrite!
  - Build property API's that expose an intuitive interface
  - QML is great for encapsulating new types and elements
  - Make sure UI elements correctly use anchors and property bindings, hardcoding pixel values inside dynamic UI elements is ***evil***
- Rewrite the top-level layer of your UI
  - E.g. the main layout of the UI (think your main.qml)
  - The navigation paradigm

NOKIA

Specific challenges and solutions

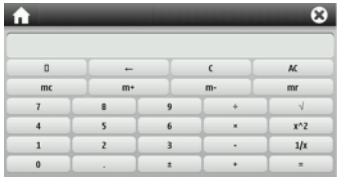
- Multiple resolutions
  - Use QtMobility (or natively exposed from C++/QPaintDevice) DPI info to determine which graphic assets to load and which .qml top-level files to use.
  - You have to have graphic assets pre-rendered for different DPI's / Resolutions.
  - SVG is not an option, because they do not render to pixel perfect versions, and SVG is too slow
- Changing graphics based on language
  - Use QML's built-in i18n features to switch Image source path's depending on language; source: qsTr("image_base.png")
- Adapting to orientation changes
  - Use Qt Quick component's Window abstraction to bind your UI states to the UI orientation. This translates in practice to either a state per orientation, or anoter version of your QML file(s) per orientation – depending on App complexity

NOKIA

Scalability Choices

- Monolithic Scalability *Frameworks* of yore chains users and platform
  - Automagically modify geometry, breaking intended design
  - Remove effort (and control) from the user to target many UXs

- QML enables scalability via anchors and bindings, allowing today;
  - Best-effort scalability by App developer
    - Choose minimal complexity that works, between orientation and sizes
    - Simple scalability at the cost of increased complexity of UI code
  - Pixel perfect design for a single resolution
    - Designed for specific resolution and Form Factor
    - Full freedom within bounds, at the cost of porting effort



- QML aims to improve scalability experience iteratively

NOKIA

How to create UIs in Qt SDK 1.1

For different resolutions and positioning of QML over QWidgets for Mobile

- How to change graphic asset depending on language?

```
Image {
   source: qsTr(“flag.png”)
}
```

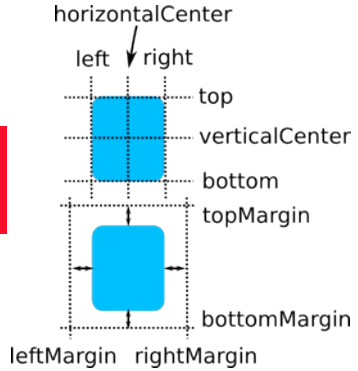
- How to get UI controls in QML without QWidget’s?
  - Write own UI controls based on QML elements
  - In the future, Qt Quick components will provide QML widgets
- How should I handle different navigation paradigms?
  - Qt and QML will never support an arbitrary number of App paradigms
  - Write a top/-level entry-point for you Application for each target domain
- How to tackle graphic asset loading for multiple resolutions
  - No QML solution yet, but can be achieved from C++ by e.g. setting the path where resources are loaded from
- How to determine top-level QML layout loading in multiple resolutions
  - Need to customize QML code for each resolution / family
  - Potentially, this can be set at qmake install time – picking the correct QML file
- How to handle differing DPI / area in touchable areas

```
viewer.rootContext()->setContextProperty("mm",viewer.physicalDpiX()/25.4);
```

```
Image { width: 40*mm; height: 15*mm }
```


NOKIA

Features in QML



Current QML Features

- Anchor layouts
  - Can represent more advanced layouts than traditional tabular layouts, while remaining flexible
- Property bindings
  - Any Item property can “bind” to another items’ property or expression thereof
- Translation support
  - E.g. to load different Image source path’s depending on current Language
- Orientation change
  - QMLviewer has code to listen to orientation changes, this can be repurposed. Near-term Qt Quick components will provide a window abstraction with Orientation, and long-term Qt Mobility will provide orientation bindings for QML

Planned QML Features

- DPI binding in Qt Mobility 1.2
- UI Orientation binding in Qt Mobility 1.2
- RTL layout support in QtQuick 1.1

NOKIA

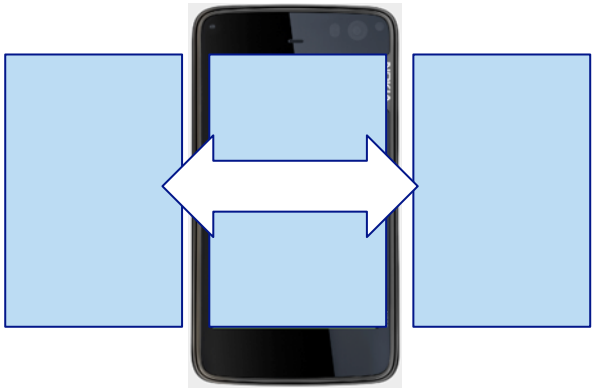
Case-study; Drilldown

- http://chaos.troll.no/~hhartz/Scalability.zip
- Problem; Create a navigation model for an email viewer that works across multiple form-factors
  - Mailbox Folders
  - Messages
  - The Message
- Invariants
  - Application logic
  - Data model
  - Low-level UI elements
- Variables
  - Resolution; n900, 5800, N8
  - Physical DPI
  - Orientation; Portrait / Landscape
  - Application State; Folders and Accounts, Message list, Message View
  - ➢ 3 state dimensions

NOKIA

Single-source approach

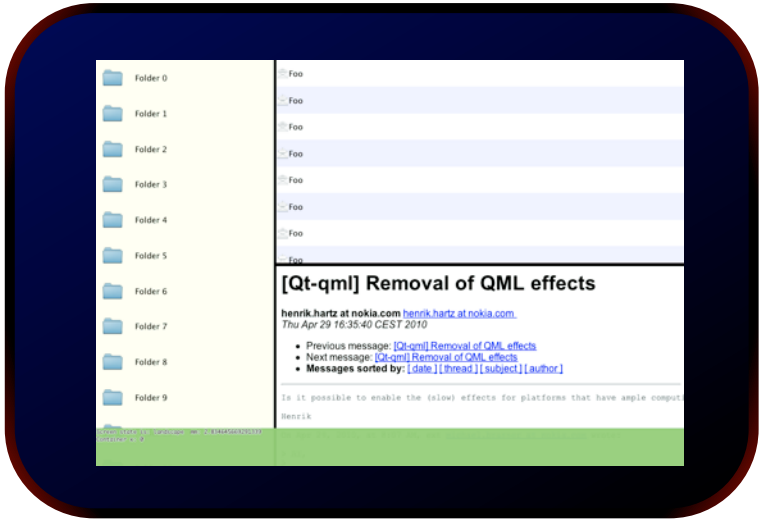
- Limiting in variability across form-factors
  - E.g. using a ‘strip’ that’s panned depending on App state
  - Only the simplest of navigation models can be expressed
- Source code becomes difficult to maintain
  - Change for one Form Factor impacts others
  - Merging variants causes exponential complexity
  - State dimensions multiply
    (resolutions * orientation * app state)
  - Behavior of individual UI elements depend on these
  - Code becomes *ugly* and *difficult to maintain*
- These were found during a simple use-case investigation
  - Problems will multiply beyond reason for a production-level App



NOKIA

Multiple-source approach

- So, what if you want the UX on your embedded target?
- Use a new ‘entry-point’ for each platform variant
  - Control which QML entry-point to load from C++
- Reuse UI components, business logic, data, states
- Differentiate in how UI is laid out and behaves
- Reduce impact of adapting to new form-factors
  - Prevent need to refactor existing code
  - Prevent unintentional UX bugs



NOKIA

Summary

- For very simple use-cases and UX'es, a scalable UI is possible to achieve
  - At the expense of code readability and development cost
  - With limitations to layout capabilities
- For more complex Apps individual UI's should be written
  - This might mean mainly a new 'top level'
  - Most of UI elements can and should be reused
  - Business logic and data is the same
  - Enable greater degrees of freedom in User Experience differentiation

NOKIA

The End

NOKIA