Fabien Decret - Adeneo Embedded

SÉMINAIRE DEVELOPPEMENT D'IHM EMBARQUEES

Summary

- Introduction
- Processor's architecture choice
- Architecture and applications' division
- Graphical Frameworks for MCU
- Graphical Frameworks from Semiconductor company
- Graphical Frameworks for high-end architecture
- Hybrid solutions
- Comparisons of the solutions
- Feedbacks
- Graphical components library
- HMI applications' samples
- Conclusion

Introduction

- Quality of the user interface is a key factor for embedded devices
- Daily use of screen forced to produce :
  - Effective products
  - With a more spontaneous and friendly surfing experience
  - Adapted to the context
- Smartphones and the use of ergonomics applications become references and highlight the importance of user-centric-design and application's ergonomics.
- This approach must be present in all industrial design process and not only in consumers' applications

HMI

- The outputs
  - 7-segment display
  - Leds, LED matrix
  - LCD screens
    - Font size and number of characters limited
    - Specific fonts
    - Colors, pixels, …
  - Sounds
    - Loudspeakers
    - Programmable sounds
- The inputs
  - Buttons (reset…)
  - Keyboard
  - Touchscreen
  - Joystick, mouse…
  - Specific inputs
- Choice
  - Connection interface (local or remote)
  - Size and weight
  - Physical restraints (temperature, vibrations)
  - Price
  - energy consumption

CAP'TRONIC

Embedded HMI

- Development constraints related to embedded software
  - Limited screen size
  - System specific
  - Look at screen once
  - Interact with only one application
  - Multi-task : only one application active, contexts management
  - Unambiguous
  - Be concise
  - Display feedbacks – Time indicators…
  - User centric

Processor's architecture choice

	MCU	OS embarqués riches	PC
Example Devices	Aux displays, Health Monitoring, Remote Controls, Robotics	GPS Handhelds, PDAs, Automotive, Set Top Boxes	Thin Clients, ATMs, Kiosks, Gaming, Industrial controls, Test Equipment
Device Features	Connected, Small, Wearable, Graphical UI	Connected, Graphical UI, Server, Browser, RAS, DirectX	PC-like performance, PC networking
Footprint	128K and Up with variable functionality supported Managed code	~1Meg + Depending on features	40Mb + Depending on features
Power	Very low power	Low power	Mains power
CPU	ARM7, ARM9, Cortex M3, Cortex M4 MMU Not Required	ARM11, Cortex A8, Cortex A9, Cortex 15, X86, MIPS, SH4, with MMU Required	X86
Screen	VGA and less	VGA and more (RGB, LVDS, HDMI and Display Port)	No constraints
Real time	Not Deterministic	Hard Real-time	Real-time capable through 3rd party extensions
HMI	emWin, eGUI, STM32 Embedded Graphic, Java ME	Qt, Microsoft, Java, Mono, Xamarin, SDK Android, hydrid architecture	MS, Qt, JAVA, Mono…

Example devices



















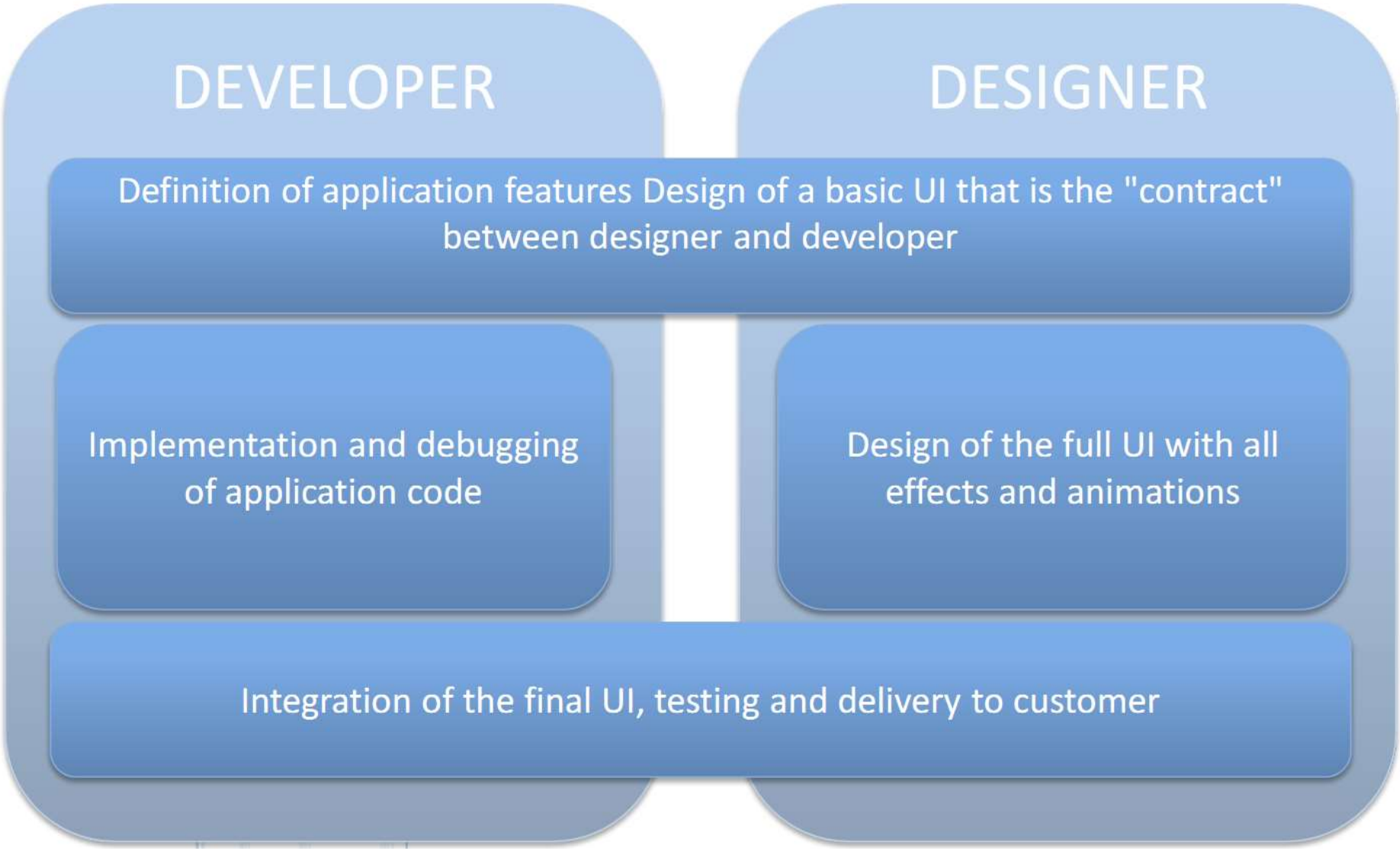
Methodology

Draft

Sketch

IHM

Methodology



Applications' architecture

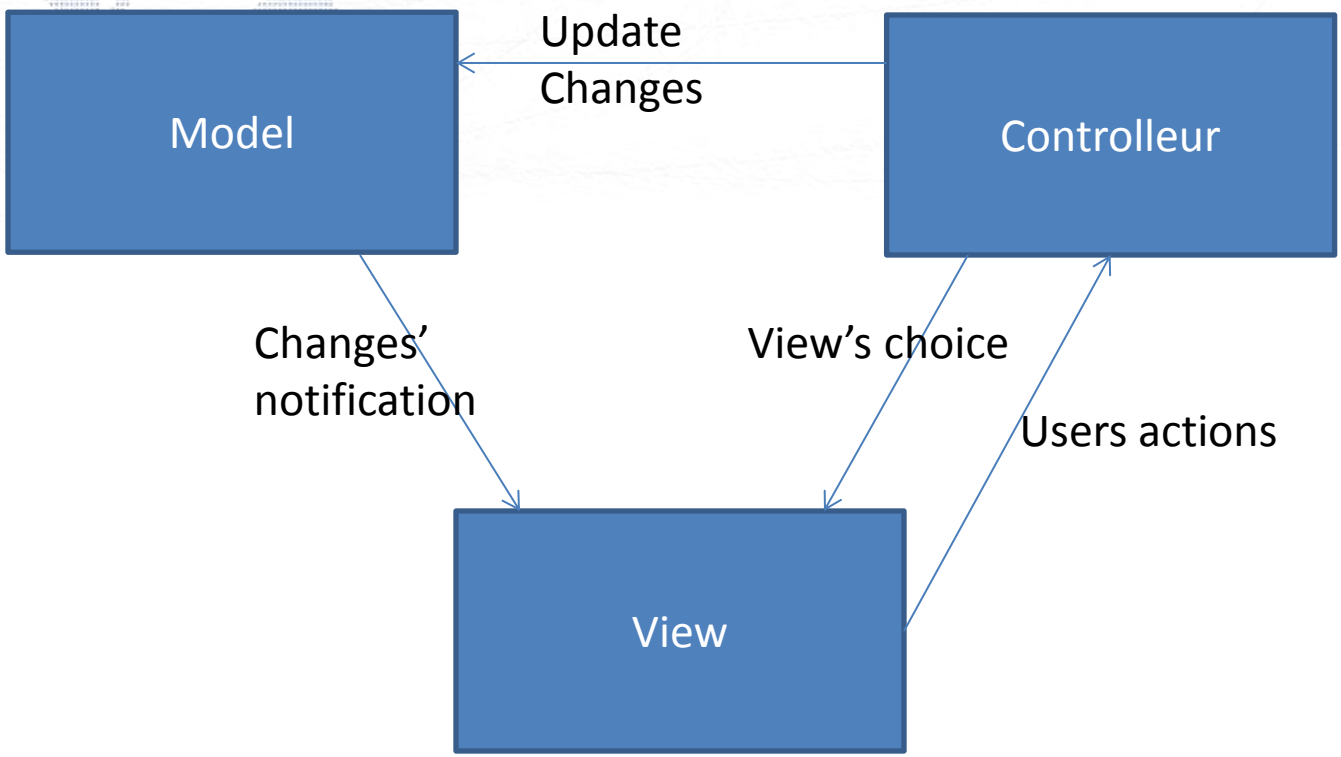
- Separation between UI and business layer
  - Designer/graphiste => UI
  - Developer=> applicative logic

- Design patterns
  - MVC
  - MVP
  - MVVM

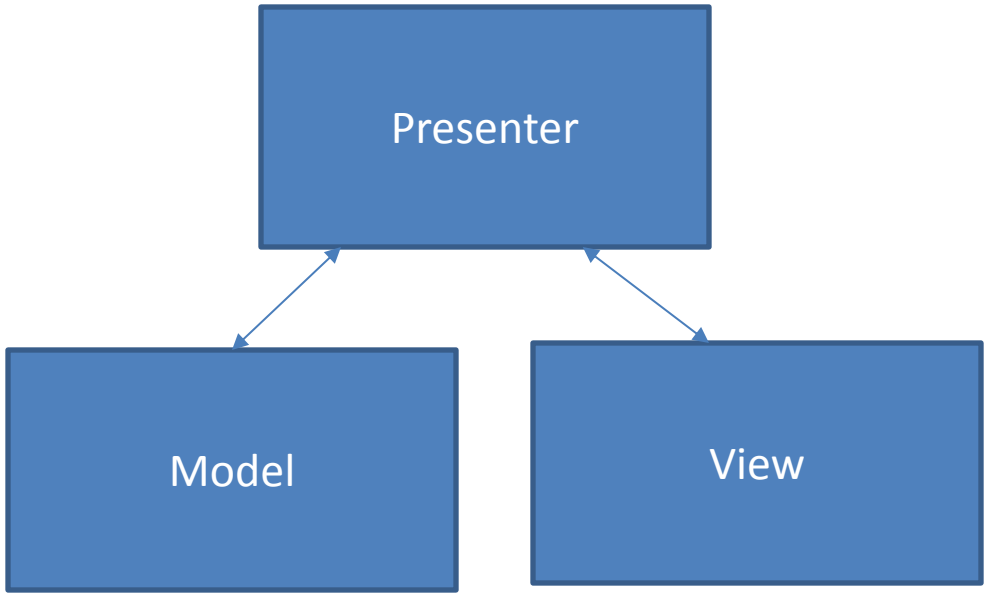
MVC

- Model View Controller
  - Division in three entities :
    - **M**odel represents the data of the application
    - **V**iew corresponds to the HMI. There can be many views implemented the data of the same model
    - **C**ontroller is responsible for exchanges between the view and the model

MVC

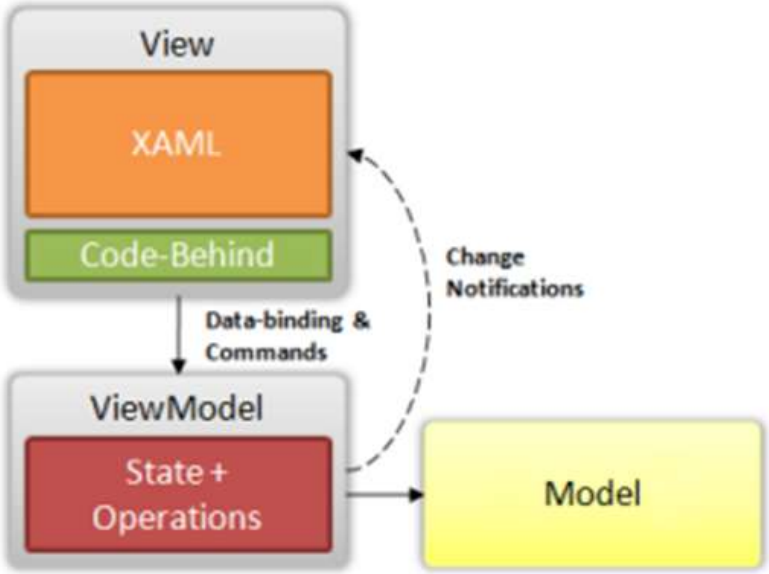

Model
Update
Changes
Controlleur
Changes'
notification
View's choice
Users actions
View

- Model View Presenter
  - Derivative from the pattern MVC
  - Remove interaction between the view and the model realized via the presentation layer, which organize the data to display in the view

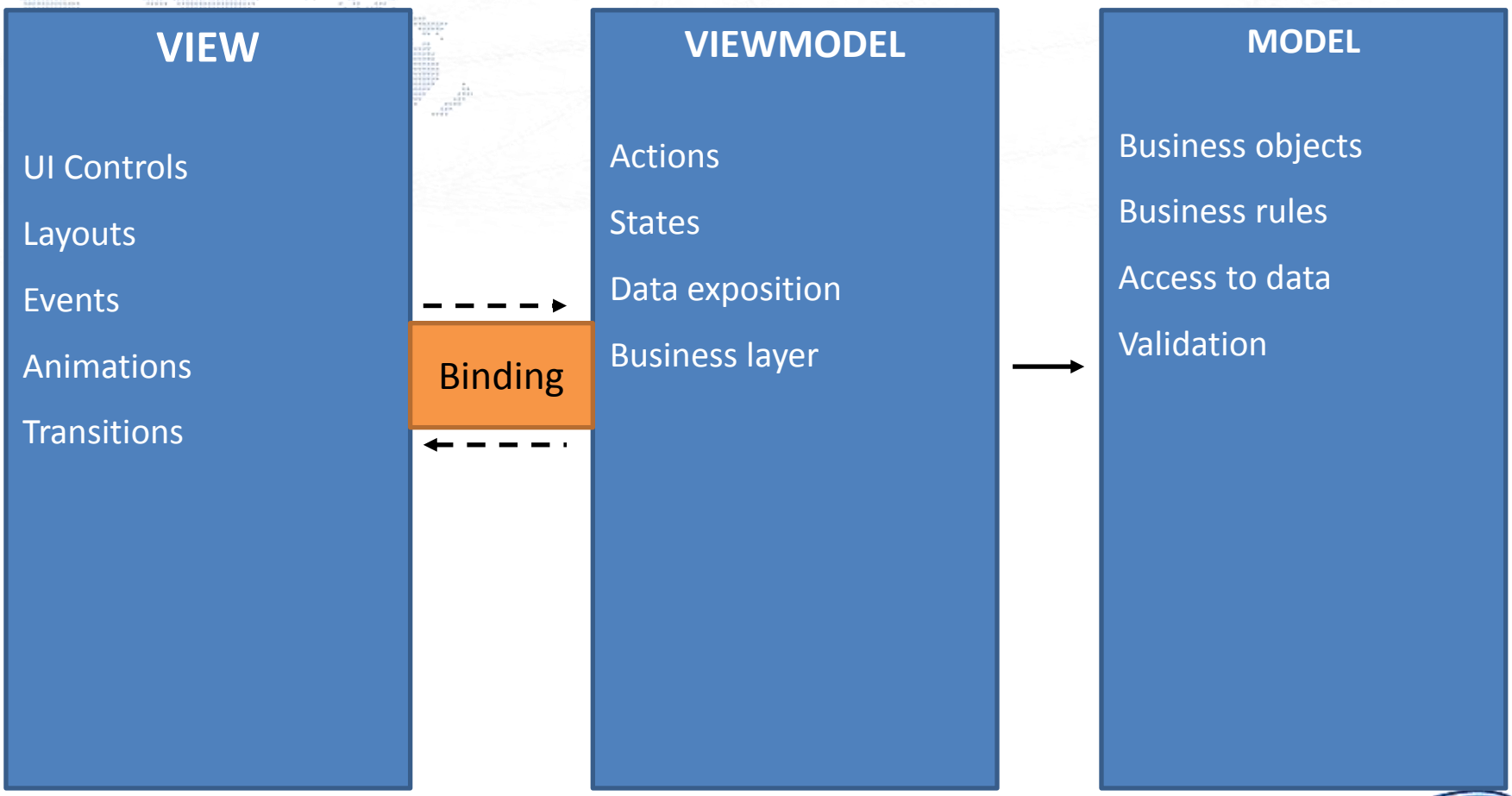


MVVM

- Derivative pattern from MVC
  - Clearly separate business layer from the HMI
  - Technology independent
  - Recommended by Microsoft WPF architects
- Best :
  - Maintainability
  - Scalability
  - Reliability…



MVVM


VIEW
UI Controls
Layouts
Events
Animations
Transitions
Binding
VIEWMODEL
Actions
States
Data exposition
Business layer
MODEL
Business objects
Business rules
Access to data
Validation

Frameworks choice

- Use of graphical frameworks and libraries
- Objectives 
  - Facilitate the implementation of the business features
  - Capitalize on know-how
  - Improve the productivity of the developers
  - Uniform application's developments
  - Facilitate maintenance and scalability

The interest in using a Framework has already proven its worth in the past.

The best difficulty is to choose the appropriate Framework !

CAP'TRONIC

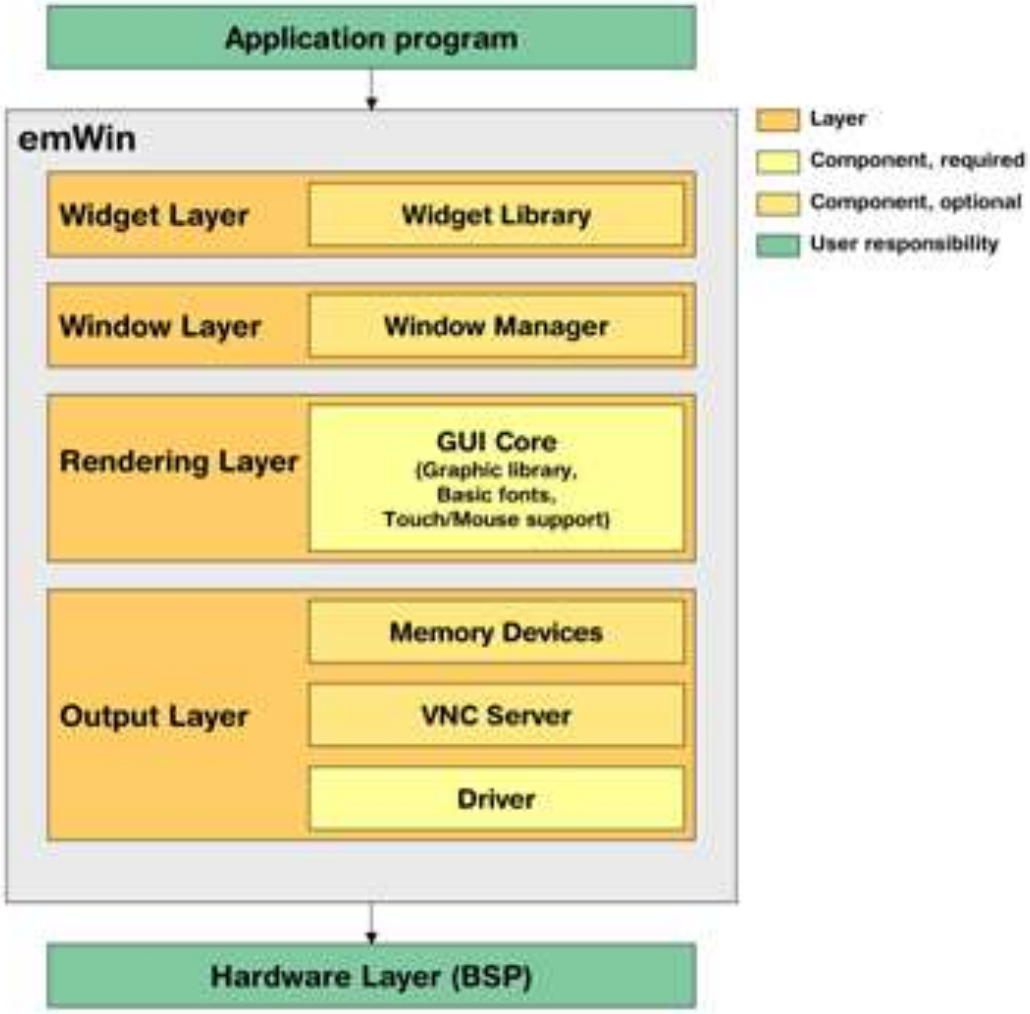
emWin (Segger)

- Description
  - Framework designed to provide efficient, processor- and LCD controller-independent graphical user interface (GUI) for any application that operates with a graphical LCD

- Supported platforms
  - Any 8/16/32-bit CPU; only an ANSI "C" compiler is required.
  - Compatible with single-task and multitask environments, with a proprietary operating system or with any commercial RTOS..

- Features
  - Small memory footprint (RAM: 334 Bytes -> 2 KBytes ; ROM: 6 KBytes -> 30 KBytes)
  - Simulation library for WIN32-Environments (all routines are 100% identical to your embedded application)

Architecture

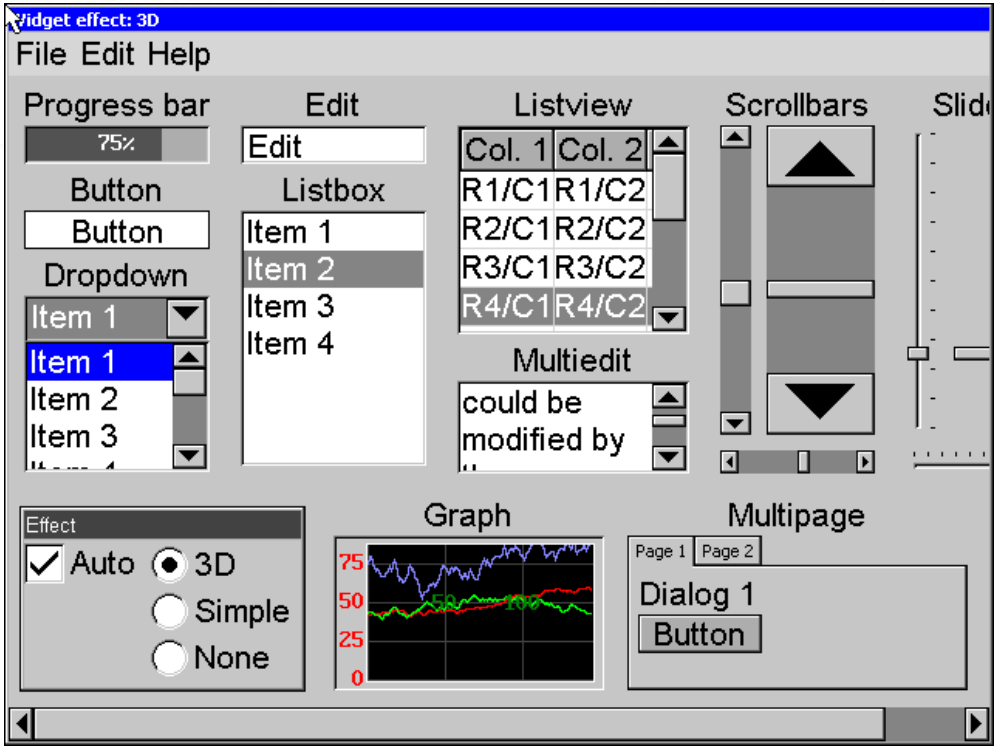


emWin (Segger)

- Licensing model

Description	Price €
emWin PRO	9800 €
emWin BASE color	4980 €
emWin BASE grayscale	3480 €
emWin BASE b/w	2480 €
emWin VNC server	1480 €
Driver	1100 €

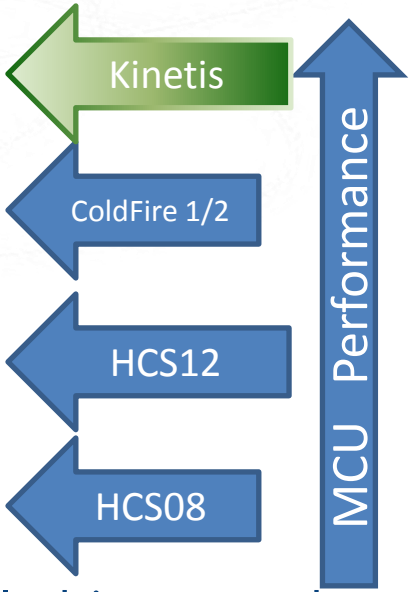
Components samples



Simulateur emWin

Freescale eGUI

- Description
  - eGUI (D4D)
  - Lightweight graphic driver
- Supported platforms
  - Low system requirements (memory, CPU, etc)
  - Easily ported to any Freescale MCU or MPU
- Footprint
  - Worst case is about 54KB Flash / ~194 B RAM (all objects used, round corners are enabled, bitmap decoders)
  - Same project without round corners and some bitmap decoders is 45KB (still all objects)
  - Typical footprint – 45KB



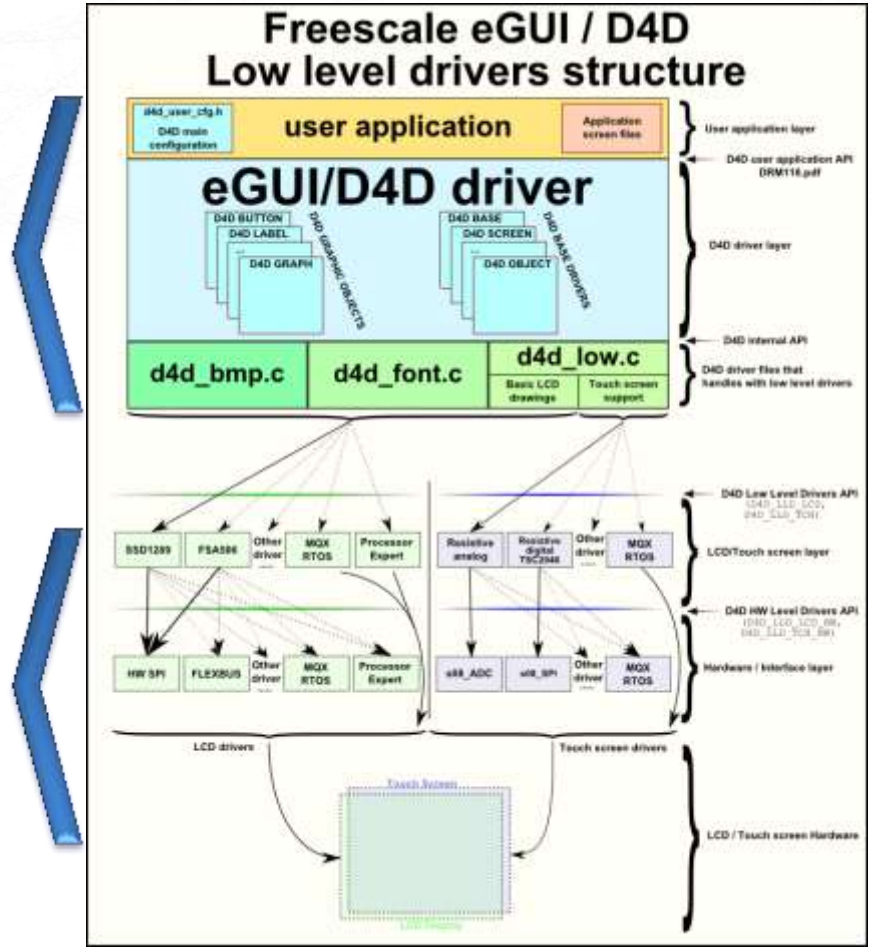
Freescale eGUI

- Features
  - Support for graphical color LCD displays
  - C language
  - Very smart support screen oriented structure of user code
  - Compounded objects (objects can be owner of child objects)
  - Embedded graphic objects (widgets)
  - Round corners support
  - Touch screen support
  - Multiple and External font support (UNICODE)
  - Transparent color / various types of color systems
  - Open source
- Development environments

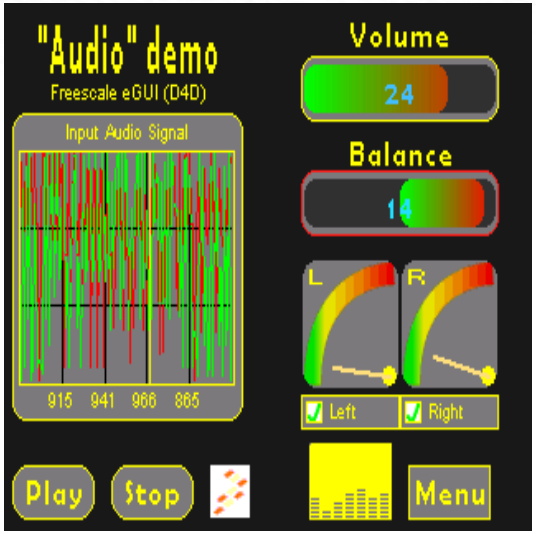
Architecture

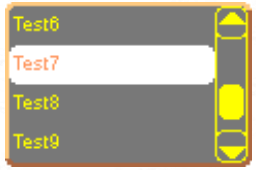
eGUI High Level

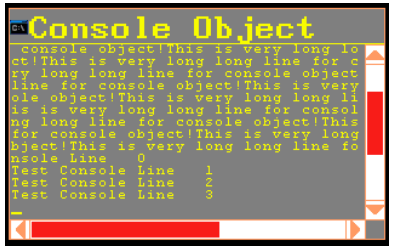
eGUI Low Level
(HW interaction)

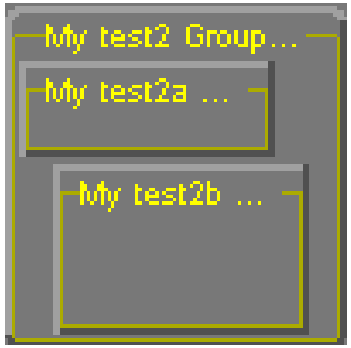


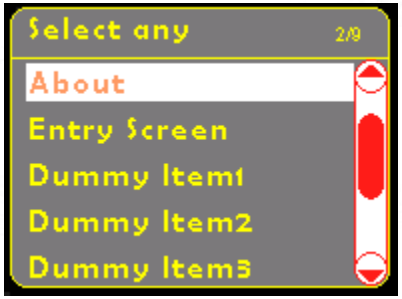
Components samples

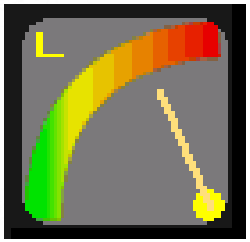












CAP'TRONIC

Swell PEG

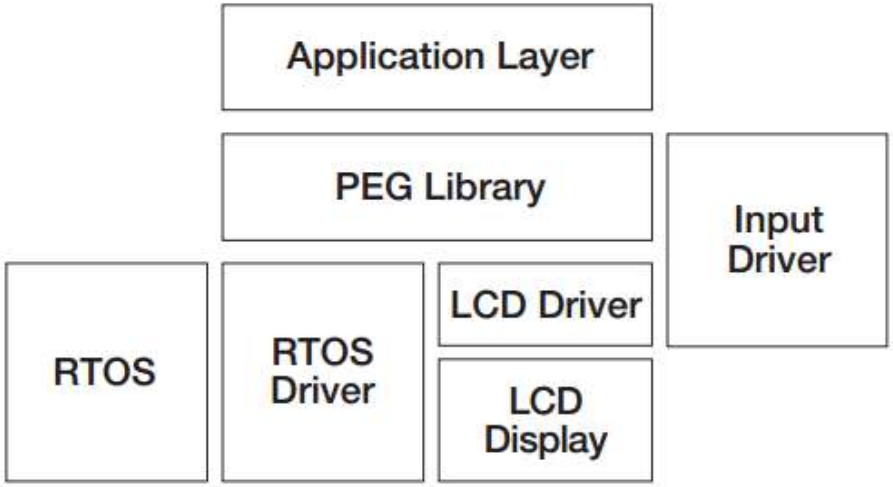
- Description
  - PEG Pro, PEG Plus, PEG Lite
  - Advanced graphics features
- Supported platforms
  - Freescale MQX software
  - Mentor Graphic® Nucleus®
  - Express Logic ThreadX
  - Green Hills Software INTEGRITY
  - Micrium µC/OS-II
  - Can be integrated with any RTOS
- Footprint
  - PEG Lite(Typical 42–52 KB)
  - PEG Plus (Typical 48–72 KB)
  - PEG Pro (Typical 64–96 KB)

Swell PEG

- Features
  - Simulation environment for PEG Lite, PEG Plus and PEG Pro for desktop
  - Provides custom drivers for most LCD panels and controllers, RTOS systems, touch screens and other input
  - Multilingual support, including Unicode
  - High color, including true anti-alias line and font drawing support and per-pixel alpha blendingScreen
  - Screen transition effects: slide-in, wipe and fade
  - Support for multiple graphics layers
  - Button, sliders, scrolling text, dials, progress bars, multiline text box and spreadsheet
- Development environments
  - PEG Window Builder for Rapid Development - WYSIWYG
  - Integrated font creation and image conversion utilities

Swell PEG

- Architecture



- Licensing model

Description	Price €
PEG Lite (10K runtine License) Up to 16-bit color, Basic widget, Dual language	6000 $ (Free on Freescale Silicon)
PEG Plus (10K runtine License) Alpha-blended images, multi-window, Custom widget integration, multi-language	7000 $
PEG Pro (10K runtine License) Screen transitions, anti aliasing, transparent text, gradients…	7500 $

CAP'TRONIC

PEG Medical

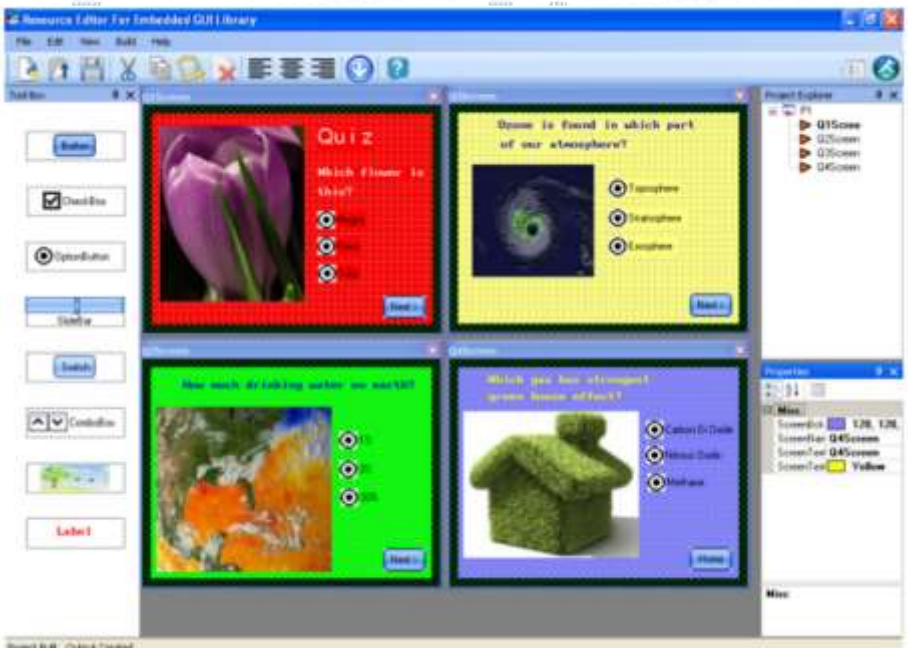
STM32 Embedded Graphic

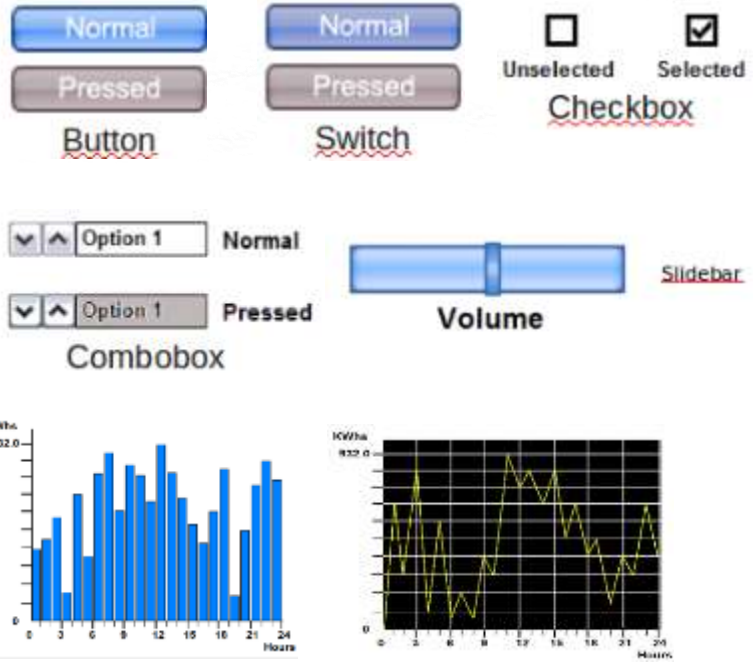
- Description
  - Library which contains a collection of routines, data structures, and macros covering the main features of a graphic library and supporting a HID device to interact with the graphic objects (touchscreen, joystick, and pushbutton).
- Supported platforms
  - Any CPU, 8/16/32-bit, to guarantee the maximum portability of any architecture or LCD controller
- Footprint
  - 64 Kbytes of Flash memory are needed
  - For a typical application, 32 KB of RAM are required

STM32 Embedded Graphic

- Features
  - Code is developed in 'ANSI-C', the code architecture follows an OOP (object oriented programming) approach.
  - The library has been developed and tested on an LCD panel of QVGA resolution (320x240)
  - The architecture improves the code re-usability splitting the application programming interface code from the hardware abstraction layer code
- Development environments
  - Embedded GUI resource editor application : interface design and generation of C code targeting the embedded GUI library in a few mouse clicks.
  - The library was developed supporting Raisonance Ride Kit ARM, IAR the IAR EWARM, Atollic TrueSTUDIO, Keil MDK-ARM , TASKINGVX-toolset for ARM Cortex-M3 and the related workspace/project files are included in the delivered package.

Components samples





Embedded ressource editor GUI

Application Framework Qt

- Description
  - Cross-platform application and UI framework
    - Qt 4.X
    - Qt 5X
- Supported platforms
  - Windows, Mac, Linux
  - Windows CE, Linux Embedded
  - Android, iOS, BlackBerry , Sailfish, Tizen

Application Framework Qt

- Features
  - GUI
  - Other APIs: XML parsing, Database access, File handling, Internationalization support
  - Graphics hardware acceleration
  - Development tools

Developping with Qt

- Qt language
  - C++
  - Bindings for Python, Java, PHP, …
- Two GUI development profiles
  - Qt native widgets (C++) : simple interfaces with buttons and forms
  - Qt Quick : advanced graphics and animations using the QML description language

Backend renderers

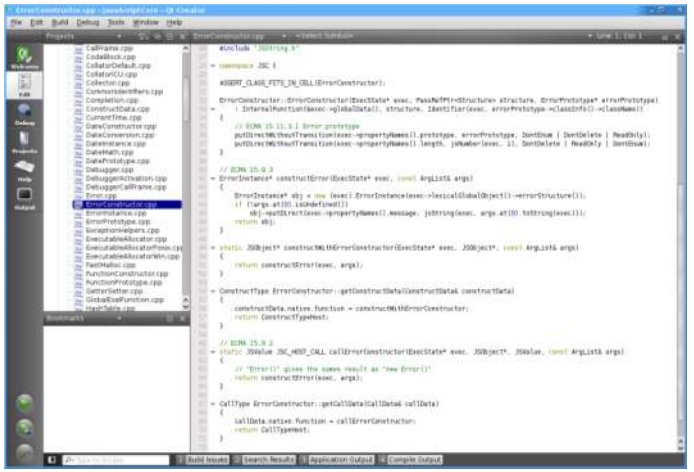
- Different backends to render graphics on the target
  - Qt 4
    - On top of X11
    - On top of the framebuffer through the QWS plugin
  - Qt 5
    - On top of X11
    - On top of Wayland (replacement for X11 designed to work with embedded devices)
    - On top of EGLFS based on OpenGL rendering
    - QWS no longer available

GUI Creation Tools

- Programming
  - Hand coded GUI
- Qt Quick / QML
  - QML declarative language

Development environments

- Create interfaces with Qt Creator
  - Supports Windows, Linux, Mac OS
  - C++ and JavaScript code editor
  - Integrated UI designer
  - Project and build management tools
  - gdb and CDB debuggers
  - Support for version control
  - Simulator for mobile UIs
  - Support for desktop and mobile targets
- QtDesigner
  - Graphical tool for GUIs



CAP'TRONIC

Native widgets

- C++
  - High performances, easy and safe
  - Entirely customizable
  - Desktop oriented, native platform look
  - Time-saving classes and algorithms

Qt Quick

- Qt Quick
  - Accessible, fast development
  - Fluid and modern UIs oriented
    - Touch, Animations, Connected
  - Completely separate UI from business logic
  - Declarative language
    - QML, Javascript

CAP'TRONIC

What is QML?

- Declarative language for User Interface elements:
  - Describes the user interface
  - What elements look like
  - How elements behave
  - UI specified as tree of elements with properties

Since Qt5, OpenGL hardware acceleration is mandatory for running QtQuick applications

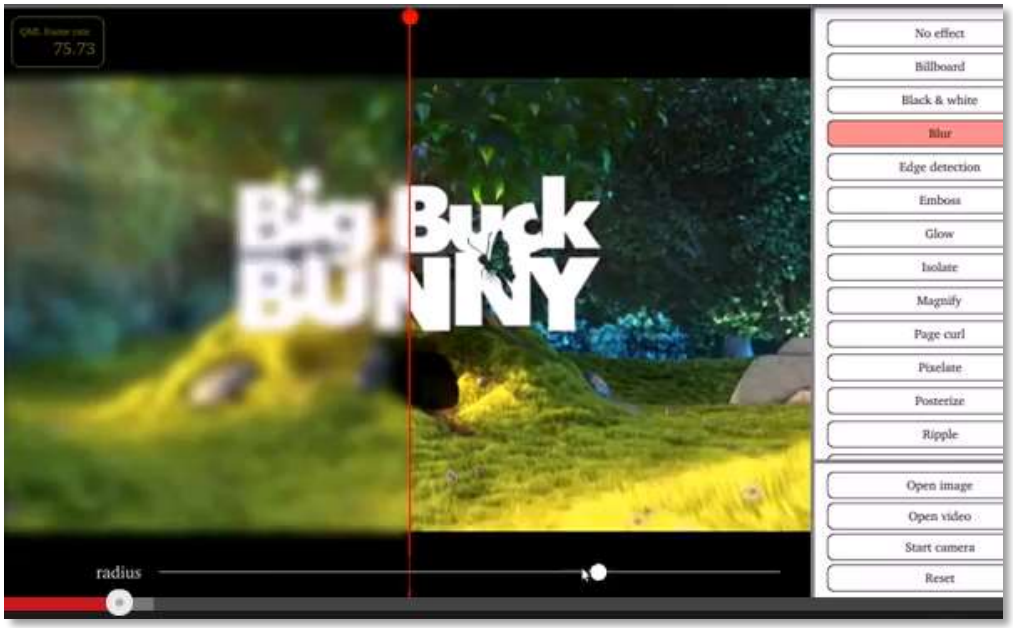
GUIs with Qt

Which one to choose ?

- Native widgets
  - No hardware acceleration, limited hardware
  - Security certification
  - “Desktop” application
- Qt Quick
  - Touch device, Gestures
  - Fluid, animated UI
  - Embedded/Mobile/Desktop
  - Fast prototyping

Graphical acceleration

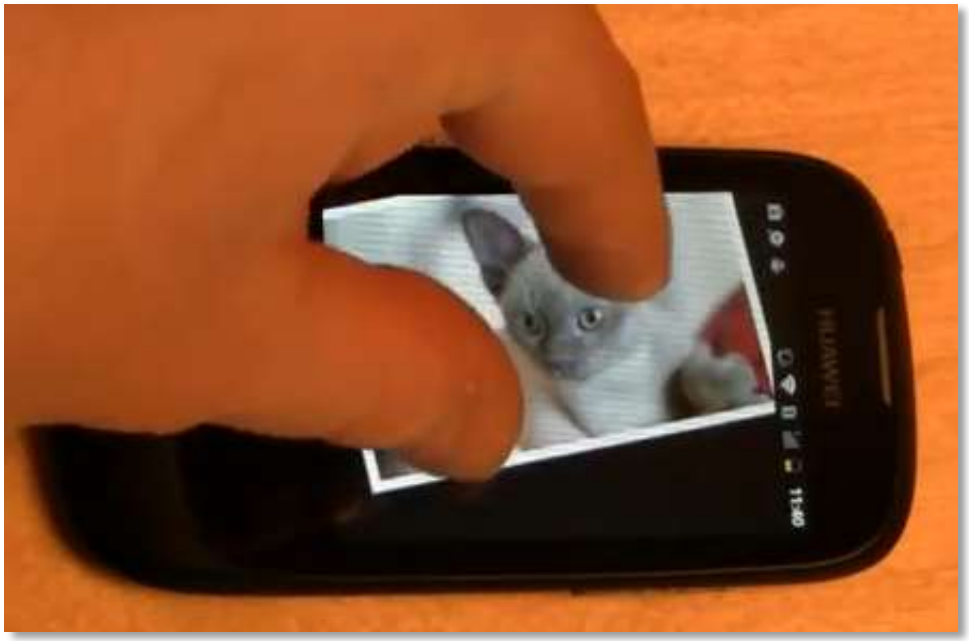
- Supports OpenGL, OpenGL ES 2, and GLSL shaders
- Allows for faster drawing, higher resolutions
- Saves CPU for other tasks



CAP'TRONIC

Gestures

- Qt Quick natively supports common gestures
  - Pinch, zoom, pan, swipe, etc.



Custom components

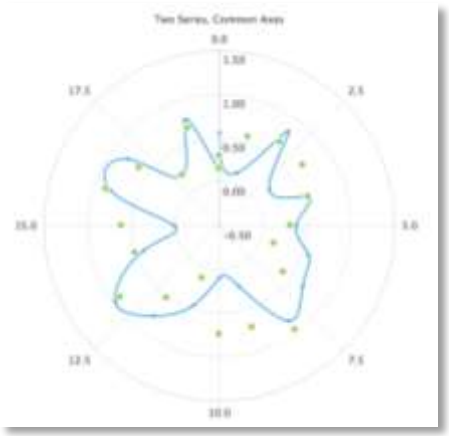
- Modify existing components
- Create your own custom component

- Native widgets and QtQuick
- Fully integrates in Qt Designer

CAP'TRONIC

And more ...

- Internationalization
- Plots
- Resources
- Testing
- XML and JSON support



- Dual licensing :
  - Qt Commercial
  - Qt Open-source

Description	Price €
Community	0
Indie Mobile Application distribution through application stores (Indie Mobile)	25€ /month
Professional Full rights to modify source code Qt support Help desk Qt & QtQuick new components Additional tools – Profiler… Embedded platforms support (Enterprise) Qt Cloud Services	149€/month
Enterprise Qt Everywhere (Windows, Linux, Mac OS X, Embedded Linux, Embedded Android, Windows Embedded, VxWorks, QNX, INTEGRITY, Android, iOS, Wi) Boot To Qt (Enterprise)	Tailored offer - Perpetual

CAP'TRONIC

QtCreator

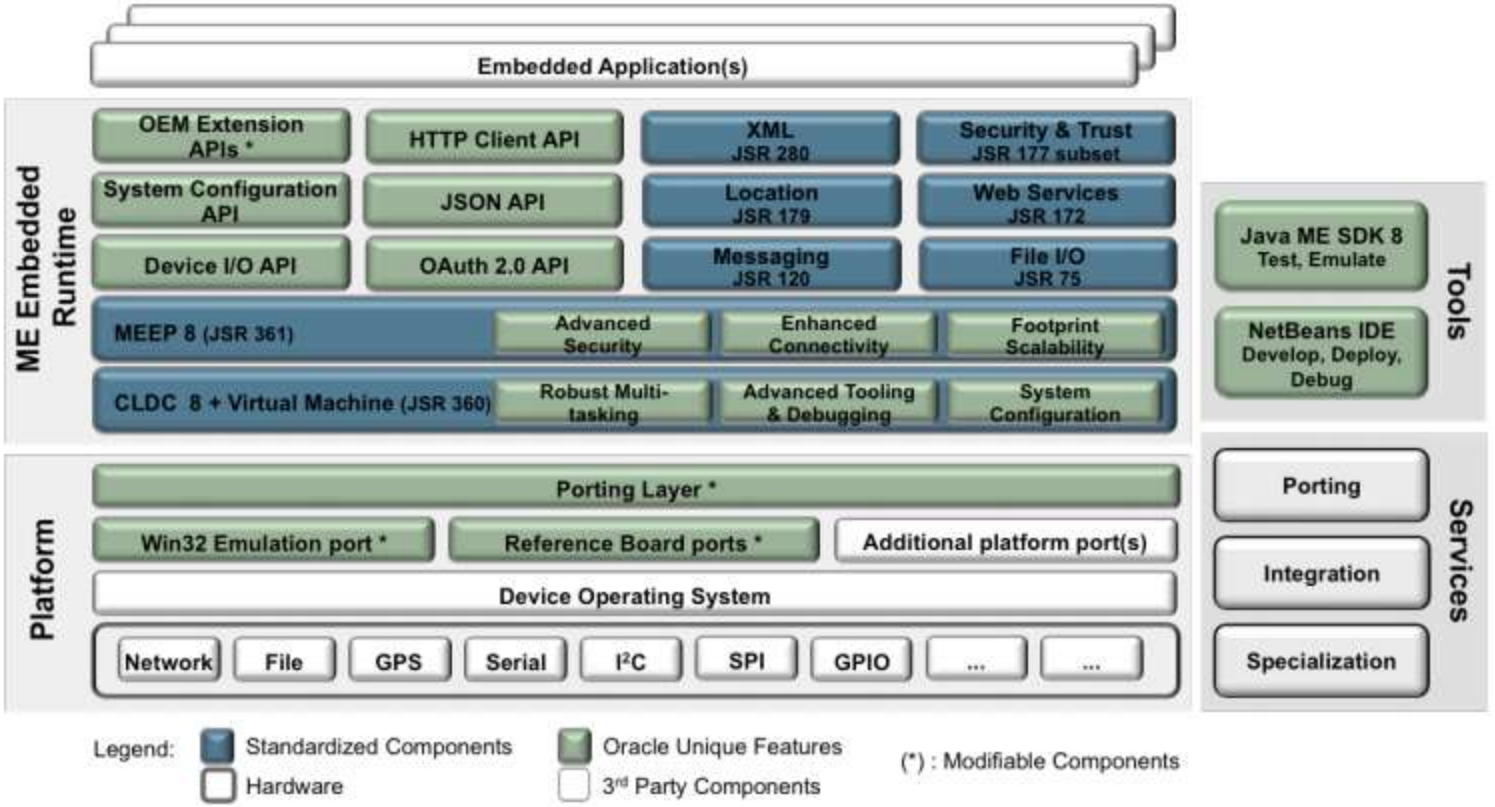
Java for Embedded

- Description
  - JAVA is a widely used cross-platform framework maintained by Oracle
  - Large framework including lots of libraries for writing headless and graphical applications
  - Works with a virtual machine interpreting Java bytecode (Java Runtime Environment : JVM + libraries & Toolkit)
  - Was developped for desktop PCs in the first place, has been since ported to embedded architectures
- Supported platforms
  - Virtual machines for embedded systems
    - Oracle provides a VM for ARM (Java SE Embedded)
    - Java SE Development Kit for ARM is now available and supported on systems based on ARM11 and more running Linux.
    - Java ME ARM9/Linux, ARM7/Linux
    - Proprietary VMs: MicroJVM, Apogee, IS2T
- Footprint
  - Reduced static footprint (60% + smaller than standard JRE)
  - Java SE : 42MB versus 140MB on X86 Linux SE 7u6 - 10 MB - 32 MB Headless, 64lMB headful
  - Java ME : Footprint requirements starting at 128 KB RAM, and 1 MB ROM

Java for Embedded

- Features
  - Java VMs for embedded systems greatly vary in terms of performance
  - Communication with specific drivers and hardware generally require writing native code through JNI
  - Platform-independent, standards-based, and efficient software environment for embedded devices
  - Device I/O APIs
  - RESTful Web Services APIs
- Development environments
  - Oracle's Java SE or ME Software Development Kit (SDK) 8, together with provided plugins for popular IDEs such as NetBeans

Architecture



Oracle Java

- Licensing

Description	Price €
Oracle Java Embedded, Binary	300
Oracle Java Embedded Suite, Binary	300
Oracle Java ME Core JSR Reference Implementation	50 000 (annual fee)

JavaFX Demo

Application Development on Windows

- Different technologies and tools can be used to develop applications on Windows
- Many different application development technologies are available on Windows :
  - Native APIs
  - MFC
  - ATL/WTL
  - .Net / .NET Compact Framework
  - Silverlight for Windows Embedded

Win32 on Windows CE

- Win32 API
  - Windows CE subset familiar to Win32 programmers
  - Some APIs are Windows CE specific: Database and Thread handling, Object Store, D3D Mobile, AYGSHELL, etc…
- API differences between CE and Desktop
  - Redundant WIN32 API functions are removed (no ASCII APIs)
  - If the function is part of an API that is not supported by Windows CE, you will have to find another solution
  - When developing a program for Windows CE, you must select the appropriate SDK and processor for the device on which your program is going to run

.Net Compact Framework

- Visual Studio 2008 .NET
  - Allow creation of Windows CE applications that use WinForms, ADO .NET, and XML Web services (Clients)
- All the languages are compiled to Intermediate Language format
  - C# application development
  - Visual Basic .NET application development
- .NET Compact framework allows rapid application development and can be integrated with native components for time-critical features
- Visual studio provides a RAD environment for managed application development, allowing easy design of forms, integration with databases and can be integrated with native components for time-critical features.

.Net Compact Framework

- Features
  - Device-side runtime support package for .NET application
  - Execution engine to manage .NET applications
  - Just-In-Time compiler for intermediate language format
  - Form-related classes, Data and XML classes, and GDI support
  - Extended subset of .NET Framework classes
  - The application is platform-independent and could easily be ported to a different device.
  - The framework provides easier access to OS services through objects instead of APIs.
  - Support of interoperability via DLL function calls or through COM type libraries.

.Net Compact Framework

- Footprint
  - The .NET runtime requires 4-8MB of storage and 2-4MB of RAM to run.
  - The runtime uses Just In Time compilation to translate Intermediate Language into machine code, this leads to less-optimized native code.
  - The Garbage Collector should suspend all managed threads during memory recovery operations and this leads to unpredictable response times.
  - Accessing some API may require additional wrappers or native code components.

Native Application Development - MFC

- Microsoft Foundation Classes were developed for the first release of MS Visual C++ under Windows 3.0.
- MFC provides wizards to design application, windows, connect events to events handler and support data validation.
- MFC provides a Document/View architecture
- MFC does not support advanced C++ features like template or multiple inheritance.
- MFC provides its own implementation of many features that are part of the standard C++ library (strings, arrays etc.).

Native Application Development - ATL/WTL

- Active Template Library/Windows Template Library
- Are based on templates and provide a thin wrapper over Windowing and COM functions.
- ATL can be used to implement COM, using objects without having to manage reference counting and manage COM basic type through wrappers.
- WTL provides support for dialog design wizards and data validation.
- ATL/WTL can be used to develop objects that can be easily integrated with .NET applications using COM interop.

Silverlight for Windows Embedded

- It's an implementation of the Silverlight engine for embedded devices.
- Allows a clear separation of UI design and application logic code.
- The UI is designed using visual tools and coded as XAML.
- The application code is written in C++, using a class framework to interact with UI elements.
- This approach will allow more independence between UI design and application development.

Windows Presentation Foundation

- WPF can be used to develop local user interfaces for managed applications on Windows XP, Windows Vista, Windows 7 and Windows 8.
- It provides 2D and 3D rendering capabilities using DirectDraw and Direct3D.

Licensing model

Description	Price €
Visual Studio 2013	639 €
Visual Studio Professional avec abonnement MSDN	1535 €

Visual Studio

C# .NET/Mono

- Description
  - The Mono project is an opensource initiative providing support for .NET on various OS
  - It implements the ECMA Common Language Infrastructure (CLI) with JIT and AOT features to speed up interpretation at runtime
  - Can display Winforms application using X
  - Compatible from .NET 1.1 to .NET 4.5, with certain restrictions
- Supported platforms
  - Linux, Mac OS X, iPhone OS, Sun Solaris, BSD - OpenBSD, FreeBSD, NetBSD, Microsoft Windows, Nintendo Wii, Sony PlayStation 3
- Footprint
  - 2 MB for a simple application + 5 MB for the mmapped libraries and assemblies
  - At this time we suggest that the minimum system memory is 32 MB
  - It is possible to reduce the footprint of the mono runtime

C# .NET/Mono

Implemented	Partially Implemented	Not Implemented

.NET 4.5

- C# 5.0 - async support
- Async Base Class Library Upgrade
- MVC4 - *Partial, no async features supported.*
- ASP.NET 4.5 Async Pipeline - *Needs an parallel processing pipeline with async support, not done.*

.NET 3.5

- C# 3.0
- System.Core
- LINQ
- ASP.Net 3.5
- ASP.Net MVC
- LINQ to SQL - *Mostly done, but a few features missing*

.NET 4.0

- C# 4.0
- ASP.Net 4.0
- ASP.Net MVC 1, MVC 2 and MVC3
- System.Numerics
- Managed Extensibily Framework - *Shared with .NET via MS-PL license*
- Dynamic Language Runtime - *Shared with .NET via MS-PL license*
- Client side OData - *Shared with .NET via MS-PL license*
- EntityFramework - *Available since Mono 2.11.3.*
- Parallel Framework and PLINQ
- CodeContracts - *API complete, partial tooling*
- Server-side OData - *Depends on Entity Framework.*

C# .NET/Mono

Implemented	Partially Implemented	Not Implemented

.NET 3.0

- WCF - *silverlight 2.0 subset completed*
- WPF - *no plans to implement*
- WWF - *Will implement WWF 4 instead on future versions of Mono.*

.NET 2.0

- C# 2.0 (generics)
- Core Libraries 2.0: mscorlib, System, System.Xml
- ASP.Net 2.0 - *except WebParts*
- ADO.Net 2.0
- Winforms/System.Drawing 2.0 - *does not support right-to-left*

.NET 1.1

- C# 1.0
- Core Libraries 1.1: mscorlib, System, System.Xml
- ASP.Net 1.1
- ADO.Net 1.1
- Winforms/System.Drawing 1.1
- System.Transactions
- System.Management - *does not map to Linux*
- System.EnterpriseServices - *deprecated*

CAP'TRONIC

MonoDevelop

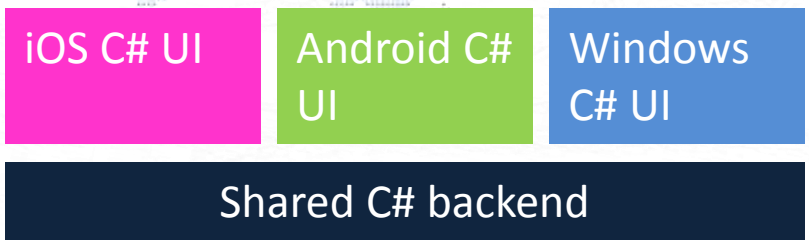
- Development environments
  - Multi-platform
    Supports Linux, Windows and Mac OS X.
  - Advanced Text Editing
    Code completion support for C# 4, code templates, code folding.
  - Configurable workbench
    Fully customizable window layouts, user defined key bindings, external tools
  - Multiple language support
    C#, Visual Basic.Net, C/C++, Vala
  - Integrated Debugger
    For debugging Mono and native applications
  - ASP.NET
    Create web projects with full code completion support and test on XSP, the Mono web server.
  - Other tools
    Source control, makefile integration, unit testing, packaging and deployment, localization, Mono Migration Analyzer tool to help port a Winforms application to Mono (Features called but missing in Mono, Calls to P/Invokes, Win32 specific feature, Calls to methods that throw NotImplementedException)

Xamarin

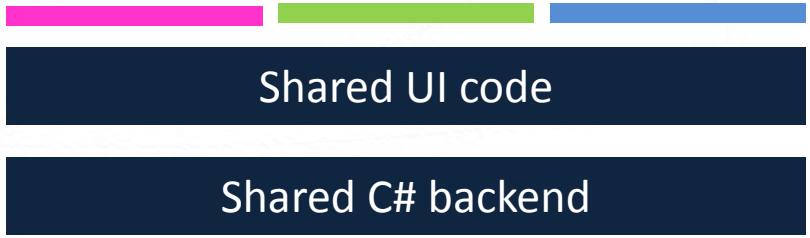
- Description
  - Build native apps for Android, iOS and Windows
  - Native application with code sharing
- Supported platforms
  - Android
  - iOS
  - Windows
- Footprint
  - small footprint (2.5 MB)
- Features
  - Covers 100% API iOS, Android, & Windows / access to native SDK
  - Compilation :
    - iOS : C# is ahead-of-time (AOT) compiled to ARM assembly language
    - Android : C# is compiled to IL and packaged with MonoVM + JIT'ing
    - Windows : C# is compiled to IL and executed by the built-in runtime
  - provides facilities for directly invoking Objective-C, Java, C, and C++ libraries

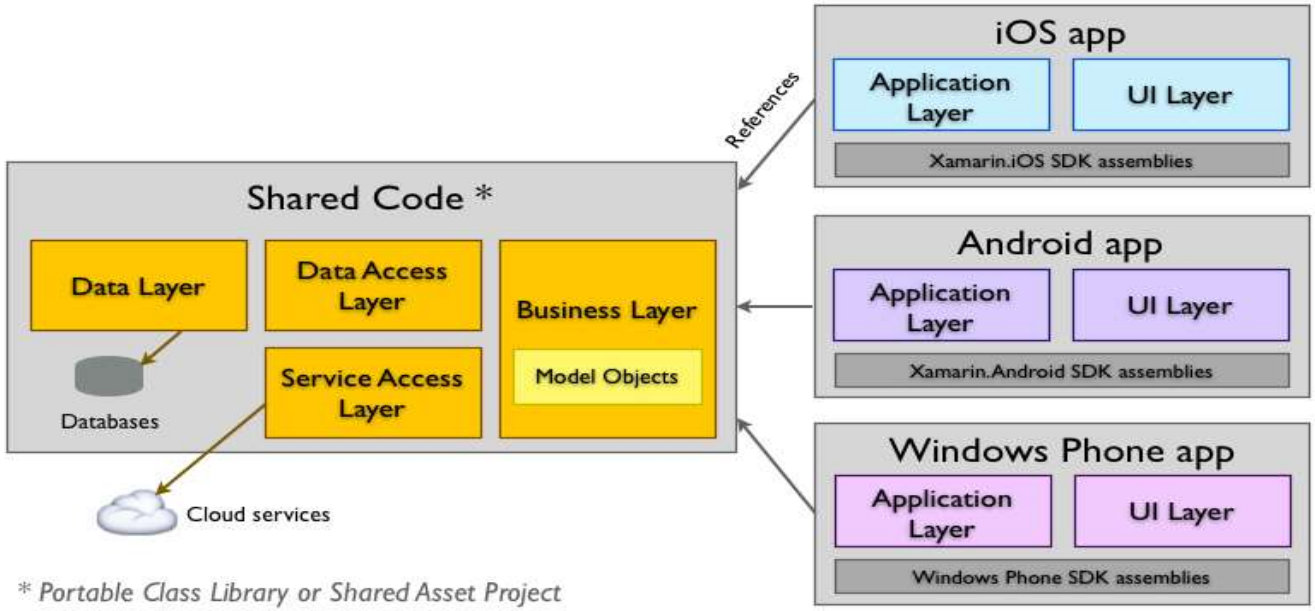
Xamarin architecture

Tradionnal Xamarin approach



With Xamarin Forms





Xamarin

- Development environments
  - Xamarin Studio
  - Visual Studio

- Licensing model

Description	Price €
Business licence	999 $ / year / platform
Professional licence (Prime components, support)	1899 $ / year / platform

SDK Android

- Description
  - Applications developed in the Java programming language using the Android software development kit (SDK)
  - Android uses JAVA language with its own VM (Dalvik/ART)
    - Not strictly JAVA compliant, standard libraries are not available
    - Uses its own bytecode format (DEX), only language is derived from Oracle's JAVA
- Supported platforms
  - ARM, x86, MIPS

SDK Android

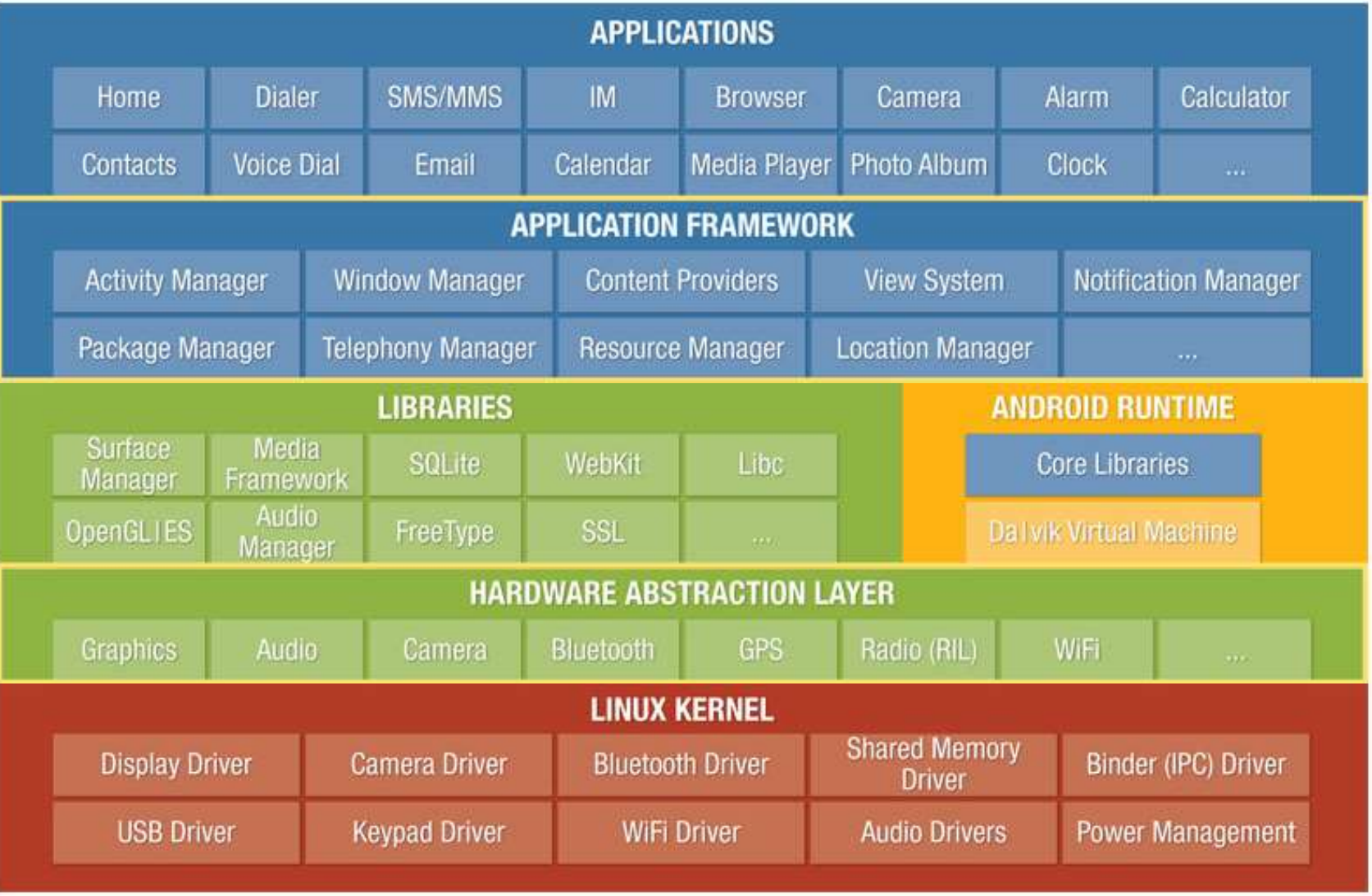
- Features
  - **A**ndroid Pac**K**age (APK)
  - SDK offers Visual Components (Activity/Fragment/Dialogs), Resources Files, Layouts, Values, Drawable, Widgets & custom views, Services, Adapters
  - Permissions' management
  - Applications are isolated, for communication there are Broadcast Receivers, Content Providers
  - SDK offers :
    - Network facilities : SMS, HTTP, JSON, Socket, Bluetooth, Wifi
    - Storage facilities : Files, SQLite, SharedPrefs, ContentProviders
  - Sensors API
  - Threads management and Handlers
  - Native Development Kit for applications or extensions in C or C++

CAP'TRONIC

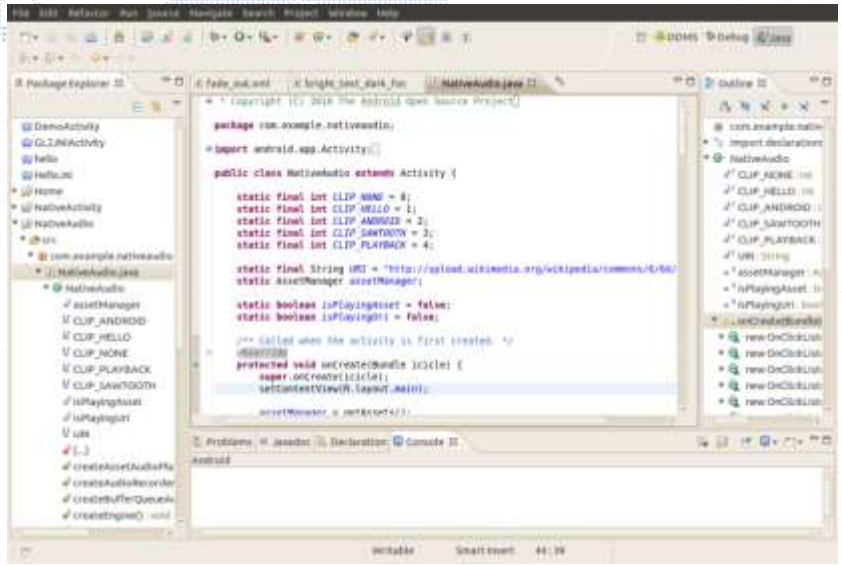
SDK Android

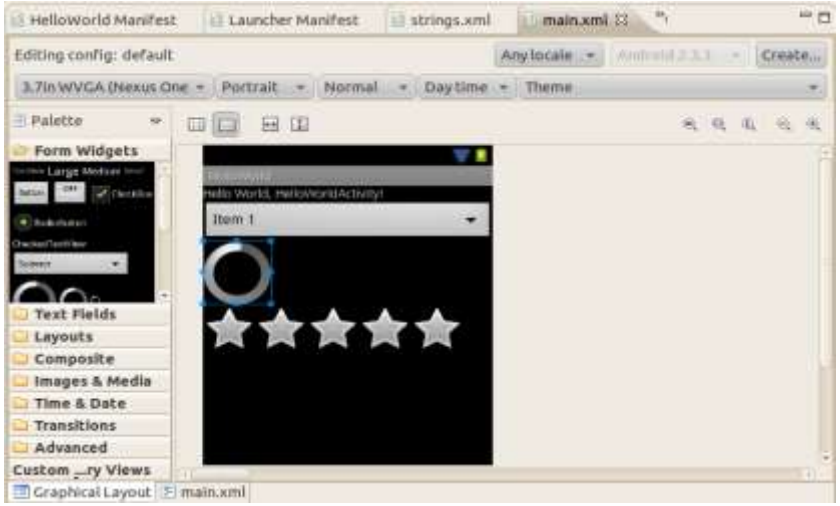
- Development environments
  - Eclipse, Android Studio
  - A comprehensive set of development tool – ADT, LogCat, DDMS, debugger…
  - Handset emulator
  - Documentation
  - Sample code

Android architecture



Tools, HMI , life cycle





CAP'TRONIC

Android SDK

Alternative projects (1)

- DirectFB: low-level graphic library
  - Runs directly on top of the framebuffer
  - Handles input: mouse, keyboard, joystick
  - Can be hardware accelerated if available
  - Provide windowing feature
- GTK: complete framework, used by Gnome
  - Standard API in C, bindings for other languages
  - Can run of top of X or DirectFB
  - Part of the larger GNOME Mobile platform

Alternative projects (2)

- Enlightenment foundation libraries
  - Very powerful, but complicated to use due to the lack of documentation
- FLTK
  - Lightweight C++ toolkit. Version 2.x works only on top of X.org. Used by relatively few applications.
- WxEmbedded
  - The embedded version of WxWindows. Works on top of X.org and DirectFB

Hybrid solutions

- Description
  - Developed with web technologies (HTML5, CSS and JavaScript)
  - Run inside a native container (webview)
  - Browser renderer : WebKit, IE10
  - A web-to-native abstraction layer enables access to low level features
  - Low-level can be :
    - Qt
    - Java
    - C#...

Hybrid solutions

HTML 5 / CSS / JAVASCRIPT
(Web browser renderer)

Embedded device

Web to native HAL

Business Engine

HAL, Drivers

Remote Access from a web browser

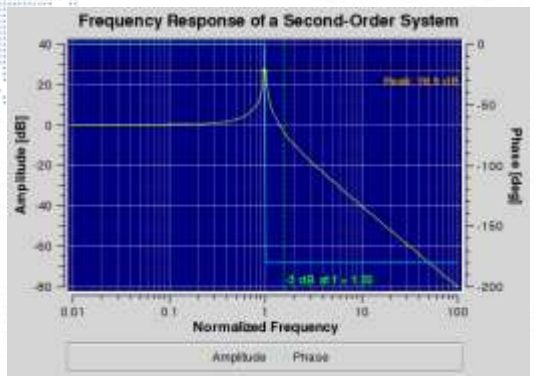
CAP'TRONIC

Crank software

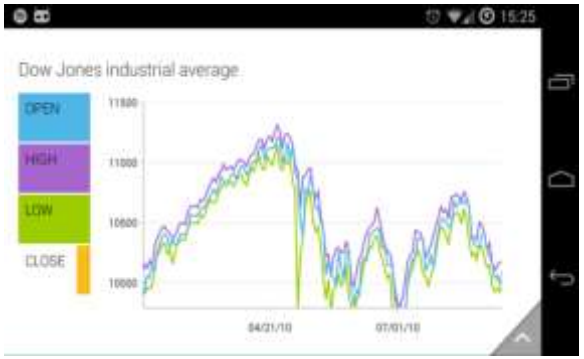
- Description
  - The Storyboard Embedded Engine is a runtime framework that allows a description of a graphical application to be interpreted and executed
  - Content is developed in Storyboard Designer
- Supported platforms
  - All embedded platforms
  - Storyboard has an embedded engine optimized for each CPU/OS/Rendering combination
- Model licensing
  - On-demand

Other graphical frameworks

- Qt
  - Qwt
  - Colibri



- Compact Framework
  - Resco
  - Telerik platform
- Android
  - Resco
  - Telerik platform
  - Graphview
  - Android plot



FEEDBACKS

Feedbacks

- Java in embedded systems
- Mono in embedded systems
- Qt
- QtQuick
- Windows Frameworks
- Android SDK
- Alternatives

Java in embedded systems

- Java SE has been designed to be run on desktop PCs from the very beginning
  - Java ME on the other hand for tiny devices, but not compatible with full Java SE
- Third-party initiatives have tried to fill the gap
  - Java support is generally only partial
  - Performance greatly vary between VMs and is generally poor
  - Such solutions are generally licensed
  - Oracle VM ported to ARM shows bad performance
- Android provides an interesting solution
  - Java language is easy-to-approach for Java-skilled developers
  - No support for standard Java libraries, making it hard to port desktop Java applications
  - Shows very good performance due to high optimization

CAP'TRONIC

Advices to embedded Java

- Some parts of an application may be ported to Java embedded, if
  - No UI is implied, UI frameworks such as Swing will most likely have bad performance. JavaFX can run on ARM platforms.
  - Not implementing CPU-intensive algorithms, which should be implemented in native code
  - It is a significant amount of code to be reused, otherwise probably better to be ported to more « embedded-friendly » technologies
- If used, special care should be taken for benchmarking Java VMs and picking the best one (performance/features/price)
- Access to specific drivers has to be written in native code

Mono in embedded systems

- Easy to integrate to your system with Yocto, thanks to the meta-mono layer
- Interesting tools for developing and porting existing .NET applications
- Not much feedback, rarely used in embedded systems
- May be used if code reuse is a big concern
  - Avoid writing UIs because of performance matters (Winforms require a X server)
  - Specific hardware access will have to be rewritten in any case
  - Run a benchmarking application on the target before jumping in
  - Time-critical code should be implemented in native code

Qt (1)

- Great framework with a lot of easy-to-use libraries to speed-up development
- Easily integrated into your system with Yocto, thanks to the meta-Qt5 layer
  - Yocto can automatically generate SDKs to be instantly used with QtCreator
- Extensive documentation
- High-performance due to native code and « embedded-friendly » backends
- Great for writing UI code in different contexts
  - Basic CPU-friendly UIs with QtWidget
  - Advanced eye-candy UIs with QtQuick
  - Designer tools to ease-up UI integration
  - Qt5 now supports multitouch input
- Very actively maintained, frequent releases
- Very powerful signal/slot mechanism for making asynchronous calls

CAP'TRONIC

Qt (2)

- Cross-platform
  - Easy to prototype your applications on a PC instead of using an emulator
  - Same code may be shared between your embedded and desktop applications
- QtCreator is a fine IDE, Eclipse or Visual Studio can also be used for legacy Eclipse users
- Accessing drivers can be done directly from Qt applications, using standard POSIX APIs
- Be careful when writing native code, no garbage collector for cleaning up memory leaks => use detection tools instead
- Becoming a de-facto standard for application development on Linux embedded

CAP'TRONIC

QtQuick

- QtQuick is a framework for advanced GUI development
  - Separates the UI from functional code
  - Description QML language is used, similar to JavaScript
  - Animations and UI code can be coded directly in QML
  - Uses extensible data models for displaying and automatically refreshing information
  - Possibility of creating new QML objects, to be instantiated anywhere for better UI coherency
  - Dynamic loading of QML files for creating application themes
- An application may be entirely written in QML, and use “qmlviewer” to be executed at runtime
- QtQuick requires hardware acceleration to run smoothly on embedded devices

CAP'TRONIC

Microsoft solutions

- Basic components are poor – but all components are customisable
- Compact Framework is Cross-platform
  - Easy to prototype your applications on a PC instead of using an emulator
  - Same code may be shared between your embedded and desktop applications
- Visual Studio is great
- Accessing drivers can be done directly from C#, using pinvoke
- Native solutions are old technologies
- Silverlight applications must be optimized

Android SDK

- Powerful development tools
- Easiness for installing the package
- Possibility to create custom and dedicated reused components
- Layouts systems for targeting multi-resolutions application
- APIs for accessing to all hardware devices and interact with system features (alarms…)
- Cartography, vocal recognition APIs included…

Alternatives

- Enlightment: performance and eye-candy applications, though complex to approach
- DirectFB: Used to be a very good solution, now getting less active and outperformed by Qt
- GTK: Definitely too heavy to be used on embedded devices. APIs are in C, leading to very complex code

EXAMPLES

Examples of HMI' applications

- Critical elements to address:
  - Architecture of the application
  - Start time is important for first impression
  - Do asynchronously the loading of large amounts of data
  - Performance on Embedded platforms:
    - Hardware components and drivers to improve performance
    - Optimize OS components
    - Optimize native code of the application

Examples of HMI' applications

- Java or C# application :
  - Desktop code application
  - Non maintainable application
  - Non performance application

➢ Use an embedded compatible framework, not too heavy
➢ Use a simple pattern MVC or MVP
➢ Use Database for embedded systems (SQLite for example)
➢ Use native code to optimize complex algorithms
➢ Fewer objects are better for performance
➢ Reflection is expensive

Examples of HMI' applications

- Silverlight application on Windows Embedded :
  - Smoothly on a VGA screen
  - Very slow on a 1024*768 even with Open Gl renderer (Load time, Animation Speed)

➢ Optimize the renderer on each components
➢ Use Bitmap cache (cached in memory so that they can be redrawn to the screen at a later time), animations…

Examples of HMI' applications

- Qt application on Linux :
  - Smoothly on a VGA screen
  - Very slow on Full HD (final resolution)

  - BSP : Check Qt compilation with Open GL is activated
  - Qt application optimizations
    - Use cache of the graphical components
    - Create of components and hide them, avoid dynamic components creation
    - Create Accelerated Open GL components
    - Optimization of the renderer code, avoid to change context during creation of a component
    - Organize your QML to avoid code duplication
    - Use light components
    - Set correct size of the component directly

Examples of HMI' applications

- Qt C++ application on Linux :
  - Lot of memory consumption
  - Bad performance
  - Bad reliability

➢ Remove memory leaks with Valgrind

➢ Optimize your C/C++ code for embedded systems

Examples of HMI' applications

- Android application :
  - No support for multi-resolution
  - No compatible with all Android devices

  - ➢ Use the layouts and relative positions
  - ➢ No hard coded position
  - ➢ Test the application with the emulator on several configurations

Examples of HMI' applications

- Hybrid application :
  - Very slow application
  - Complicated and non-maintainable application

➢ Keep the hybrid layer small and manageable

➢ Cut out the code properly

  ➢ HMI in web layer

  ➢ Business in native layer

Examples of HMI' applications

- Hybrid application :
  - One big JavaScript file on web side
  - No multiscreen resolution compatible

➢ Use a Javascript framework : Angular.JS for example that implement the MCV pattern

➢ Use JS minification

➢ Use a multiscreen CSS style sheet : BootStrap for example

Conclusion

- A lot of frameworks and tools are available
- Choice of framework is important but upstream work is also very important
  - Definition of the hardware
  - Definition of the HMI
  - Ergonomics
- Make performance a requirement and measure it
- Develop good architecture and programming habits with tips and tricks