KF6 the Architecture Overview
Time to Slice Things Up Yet Again

Kevin Ottens

Akademy 2021, June 19th
whoami

- Started to use KDE with 1.0-beta1 in 1997
- Procrastinated until 2003 to finally contribute code
- Fell in love with the community back then
- Kept doing things here and there... most notably helped with:
  - kdelibs
  - KDE Frameworks architecture
  - the KDE Manifesto
  - Community Data Analytics
- Part of the enioka Haute Couture family
- Living in Toulouse
Let’s Drive Down The Memory Lane
KDE Platform

Basically it was . . .

- kdelibs
- kdebase-runtime

With a rather monolithic development
kdeinit4 Dependencies
libplasma Dependencies
Tokamak 4, February 2010
Someone Took an Ax!
KDE Platform on Mobile?

KDE Platform Profiles

Low fat KDE Platform for your pleasure
(KDE5 free, no added sugar)
Platform 11, June 2011
Ten Years Ago!
Welcome to a Cabin in the Swiss Alps...
Someone Took a Machete!
Bunch of Geeks in the Middle of Nowhere
Kanban Board (Start)
Kanban Board (End)
KDE Frameworks 5 Dependency Plan

- Notice
  - rows
  - columns
  - colors
  - shapes
- Where is KParts? :-)

KF6 the Architecture Overview
• Looks familiar? At least the tiers should be... 
• What about the types? 
• See! Types have dependency rules as well! 
• Was supposedly usable as a maturity model
More About the Framework Types

The constraints from Types are the following:

- **Functional Qt Addons** cannot have runtime dependencies;
- **Integration Qt Addons** can have an optional runtime dependencies and aim at integrating with the underlying OS/Platform;
- **Solutions** have mandatory runtime dependencies, it is part of their design and where their added value comes from (think scalability, resource sharing, resilience, etc.).

- Sounds complicated?
- Clearly it is... it never quite picked up in conversations
- It was (clumsily) trying to draw the line between:
  - portable APIs abstracting some platform facility
  - APIs acting as building blocks to make platforms
Today
KDE Frameworks Dependency Graph

- Each layer of nodes is a tier
- Tier 1 at the bottom
Visualizing the Type of Framework

- Red: functional framework
- Green: integration framework
- Blue: solution framework
Visualizing Invert Dependency Type

- Arrows colored by the type of the framework having the dependency
- NOT the dependency itself (this is important)
Visualizing Invert Dependency Type (Sample)

- **Green dot: OK**
  - Two solution frameworks depending on an integration framework
- **Red dot: OK**
  - One solution framework and one integration framework depending on a functional framework
- **Blue dot: Woopsie!**
  - One solution framework depending on a solution framework...
  - But one functional framework depending on a solution framework?!
Something Fishy in the Types?

- Hard to understand
- Clearly didn’t play their role in the maturity system we envisioned
- One extra dimension
- Probably quite a few mistakes in the current metadata due to the above
- Can we do better?
Conflated System Abstraction and Implementation

- Let’s go back to the original intent of the types
- We provide platform abstractions for portable apps
  - In some way we provide an extended Qt
- We also make building blocks for our own platform
  - Plasma on top of X11/Wayland on top of some POSIX system
- Our KDE Frameworks product contain both
- This is unfortunate!
Behold the Plasma API

- Let’s define a Plasma API:
  - baloo
  - kactivities
  - kactivities-stats
  - kglobalaccel
  - knotifyconfig
  - krunner
  - kwallet
  - plasma-framework
  - purpose

- There are more of them released with Plasma itself
  - Out of scope here...

- Should applications use those APIs?
  - Of course!
  - The price is reduced portability though (ifdefs to the rescue)
  - This is Plasma specific
  - Similar to using Windows or macOS APIs directly
Visualizing Plasma API Parts

• Notice none are tier 1
• What is forbidden?
  – A non Plasma API framework (gray) depending on a Plasma API framework (red)
• Can you spot one? Hard to read still...
Finding Offenders: Let’s Forget the Tiers

- Better layout to visualize the issues
- Still too many edges
Finding Offenders: Keep Only Offending Edges

- Filtering out the allowed edges
- Reassuring... not too many edges left
- Still some overlaps in that layout
Finding Offenders: Keep Only Offending Nodes

- Filtering out the nodes having only allowed edges
- Not many nodes left in our graph
- Let’s finally name them!
Finding Offenders: I See You!

- Now we got our shortlist of offenders
  - KIO
  - KXmlGui
  - KDeclarative
- But why?
Finding Offenders: One Level Deeper

- KIO has kpasswdserver, it seems legit to talk to KWallet
- KXmlGui and KDeclarative both provide GUI to deal with shortcuts
Future?
Why Those Dependencies?

- Not many to dependencies to fix, this is good news:
  - kpasswdserver → KF5::Wallet
  - KF5::XmlGui → KF5::GlobalAccel
  - kquickcontrolsprivateplugin → KF5::GlobalAccel
- We likely want an even finer grained view still
- ELF Dissector to the rescue!
Someone Takes a Scalpel!
KIO → KWallet

kpasswdserver which uses:

- KWallet::Wallet::keyDoesNotExists()
- KWallet::Wallet::isEnabled()
- KWallet::Wallet::openWallet()
- KWallet::Wallet::NetworkWallet()
- KWallet::Wallet::PasswordFolder()

Easy solution: switch to QtKeychain
KF5::XmlGui which uses:

- KGlobalAccel::defaultShortcut()
- KGlobalAccel::getGlobalShortcutsByKey()
- KGlobalAccel::shortcut()
- KGlobalAccel::isGlobalShortcutAvailable()
- KGlobalAccel::self()
- KGlobalAccel::globalShortcutChanged()
- KGlobalAccel::stealShortcutSystemwide()
- KGlobalAccel::hasShortcut()
- KGlobalAccel::setShortcut()
- KGlobalShortcutInfo::KGlobalShortcutInfo()
- KGlobalShortcutInfo::componentFriendlyName()
- KGlobalShortcutInfo::friendlyName()
KXmlGui → KGlobalAccel, cont’d

Dependencies coming from:
- KKeySequenceWidget
- KActionCollection
- KShortcutEditWidget
- KShortcutsEditor
- KShortcutsEditorItem
- KXmlGuiFactory

Use cases:
- Setting/Getting global shortcuts
- Checking conflicts with global shortcuts
- Handling shortcut settings
- Reacting to global shortcut changes (since some widgets display the global shortcuts)

Potential solution:
- It already uses a HAVE_GLOBALACCEL define all over the place
  - This is a good starting point
- It misses some form of facade to abstract away the uses further
  - Adding such a facade is an option
  - Then turn it into a plugin interface
  - frameworkintegration could deliver the plugin
KDeclarative → KGlobalAccel

- kquickcontrolsprivateplugin which uses:
  - KGlobalShortcutInfo::KGlobalShortcutInfo()
  - KGlobalAccel::getGlobalShortcutsByKey()
  - KGlobalAccel::isGlobalShortcutAvailable()
  - KGlobalAccel::stealShortcutSystemwide()
  - KGlobalAccel::promptStealShortcutSystemwide()

- Dependencies coming from KeySequenceItem (KeySequenceHelper)

- Only use case: checking conflicts with global shortcuts
  - Code pretty much duplicated from KXmlGui
  - Relatively self contained

- Potential solution:
  - Provide a D-Bus call for checking conflicts
  - Looks like a common use case
  - KXmlGui could use that D-Bus interface for this use case instance of the facade
What Now?

- Tasks should be added to the KF6 workboard
  - One per unwanted dependency to solve
  - Need to settle on the solutions first

- Let’s then implement the tasks

- Profit... well, not quite...
  - That was just the technical bits
  - The real challenges are just after that bump
Someone (Gratuitously) Takes a Chainsaw!
It’s All About Communication, Stupid

- Do we move the Plasma API frameworks out of KDE Frameworks?

- Community and Release Management implications
  - Where do the Plasma API modules go?
  - Probably within Plasma
    - IMO there is value to release in sync with Plasma releases
    - Would remove a few headaches
    - e.g. the kwayland-server move is similar in a way
  - Requires the Plasma team to commit to API and ABI guarantees similar to KDE Frameworks for the Plasma API frameworks

- Marketing implications
  - How to communicate about this?
  - Obviously it is not the same messaging if it is...
Thanks You!

Questions?

ervin@kde.org
kevin.ottens@enioka.com