Declarative Retranslations in QML

Vladimir Moolle,
Integrated Computer Solutions, Inc.

Qt WORLD SUMMIT 2015

Problem statement

Given the declarative nature of QML, how does one make the UI update all translated text strings on a language change? The simple

```
Text {
    text: qstr("Translate me")
}
```

does not work this way, for example (the text stays as it was despite the language change).

There were attempts to fix this though (now abandoned, but interesting, if one is looking for insight into the topic):
https://codereview.qt-project.org/#/c/70783/
https://codereview.qt-project.org/#/c/70792/
https://codereview.qt-project.org/#/c/70793/

Usual workarounds

The (probably incomplete) list of known workarounds includes:

- emitting a “changed” signal for a string-type property appended to all translated strings, causing reevaluation of the whole binding expression
(relies on the never-documented lack of optimization / caching in QML engine):

```
text: qsTr(“Translate me”) + someObject.emptyStr // reports being “changed” on lang. change
```

- traversing all currently visible text-displaying elements from within a retranslateUI() Javascript function and reassigning new, translated values to text properties
(is more work, error-prone, may break bindings)

- reloading the whole set of currently visible items, thus forcing retranslation
(may cause screen flicker, and risks losing UI state – scroll and cursor positions, text selection, etc.)

Some insight

A) Note:

- what QML deals well with is properties (on a property change, bindings involving it get updated properly)
- if, for example, every translatable string was exposed to QML as a property, retranslations would work flawlessly...
- creating above properties by hand is out of question for most apps

B) On the other hand, a dynamically retranslated string is essentially:

- a result of a parameterized query into the backend (and ultimately, the currently active QTranslator)...
- with the only parameter being the translated text itself

This suggests a generalization...

Generalizing into a pattern

Let's imagine something named “Active Request”:

- a QObject exposed to QML...
- being parameterized by several (at least one, like with translated strings) “properties” (could be read-only or not visible to QML at all, it is enough to specify them at creation time)...
- and watching some specific events, not visible to QML (i.e. language change, in our case)

A Q_INVOKABLE could be used to create / retrieve the request instance, incl. in binding expressions, with the final “result” provided via a read-only property. For retranslations, this can be:

[a Q_INVOKABLE] **[the string-type property holding translated text]**

```
Text { text: translator.translate(QS_TR_NOOP(“Translate me”)).value }
```

[a request-generating QObject in the backend] **[the only parameter]**

This pattern nicely extends to database queries, or non-structured data aggregation (with data changing over time – just bind to a request object's property, and the changes, when they happen, will be accounted for).

More detail on the solution

Two classes could constitute a solution for declarative retranslations in QML.

A Translator (not to be confused with QTranslator descendants):

- provides a Q_INVOKABLE that creates a TranslateRequest instance (below), based on the string to be translated (ownership is passed to QML engine)
- watches LanguageChange events

TranslateRequest:

- stores the string to be translated
- is notified about LanguageChange events by the Translator and emits a changed() signal for the string-type “value” property
- returns the stored string, translated by the currently active translator, via the “value” property

This way, QML retranslations become (arguably) non-hackishly declarative. With a caveat :)

The caveat

Q: Can you guess what happens on Slide 5, when “value” property changes?

i.e.

```
                    [a Q_INVOKABLE]                  [the string-type property holding translated text]
Text { text: translator.translate(QS_TR_NOOP(“Translate me”)).value }
          [a request-generating QObject in the backend]        [the only parameter]
```

A: [on the next slide]

The caveat

Q: Can you guess what happens on Slide 5, when “value” property changes?

i.e.

[a Q_INVOKABLE] **[the string-type property holding translated text]**

```
Text { text: translator.translate(QS_TR_NOOP(“Translate me”)).value }
```

[a request-generating QObject in the backend] **[the only parameter]**

A: Obviously, due to being as brute-force as it often is, the QML engine decides to reevaluate the *whole* binding, thus calling translate() again and creating another instance of TranslateRequest (the old one keeps floating around, unreferenced, until the QML engine decides to garbage-collect it – remember, how it is its responsibility (slide 6)?)

This is more or less harmless, if TranslateRequest only translates on “value” property access (or lazily), yet, the LanguageChange events are still needlessly tracked.

The trick

To make the Q_INVOKABLE not being called unnecessarily, the binding can be split:

```
// TrText.qml
import QtQuick 2.0

Text {
    property string trText // this normally holds the QS_TR_NOOP(“...”)
    property QtObject textRequest: translator.translate(trText) // reevaluates on trText changes
    text: textRequest.value // here comes the dynamic retranslation
}
```

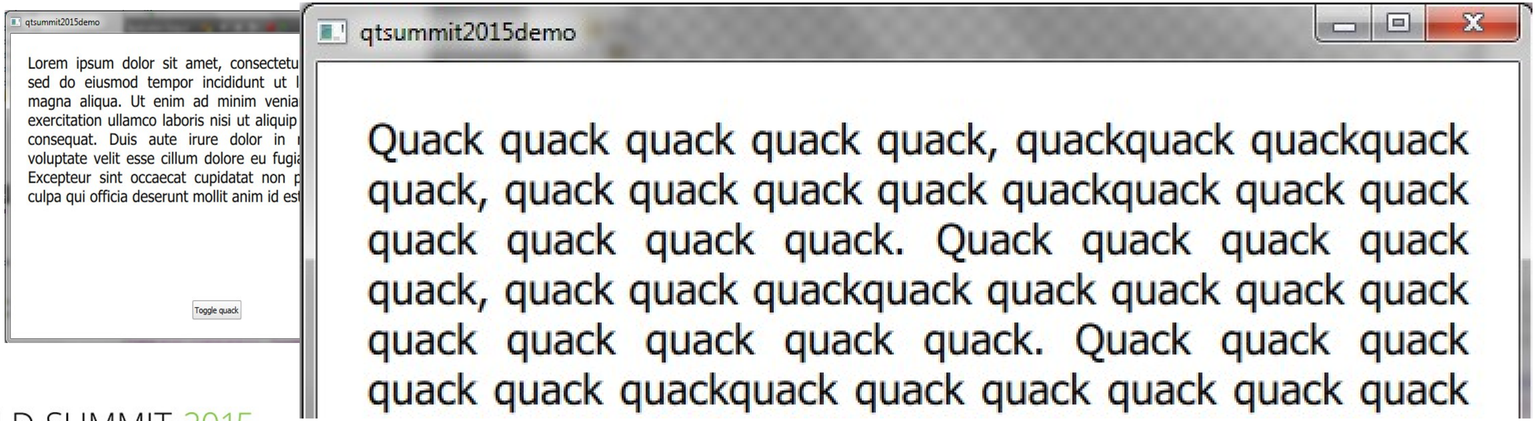
This way, a set of “retranslatable” controls can be trivially created.

Demo – meet Cute Quack

A QuackTranslator is a toy translator, translating strings into “quack language” (with relative word length, capitalization and punctuation preserved):

- “Hi” → “Quack”
- “Good morning!” → “Quack quackquack!”

An interactive demo shows a simple QML dynamically “quackificated”:



Lorem ipsum dolor sit amet, consectetu
sed do eiusmod tempor incididunt ut l
magna aliqua. Ut enim ad minim venia
exercitation ullamco laboris nisi ut aliquip
consequat. Duis aute irure dolor in r
voluptate velit esse cillum dolore eu fugia
Excepteur sint occaecat cupidatat non p
culpa qui officia deserunt mollit anim id est

Quack quack quack quack quack, quackquack quackquack quack, quack quack quack quack quackquack quack quack quack quack quack quack. Quack quack quack quack quack, quack quack quackquack quack quack quack quack quack quack quack quack quack. Quack quack quack quack quack quackquack quack quack quack quack quack

QUESTIONS? :)

Extra links

Below are some links to relevant discussions / advice:

- http://comments.gmane.org/gmane.comp.lib.qt.user/8966
- http://damagedspline.blogspot.ru/2011/07/dynamic-translations-in-pure-qml.html
- http://stackoverflow.com/questions/15355156/is-it-possible-to-change-language-on-qt-at-runtime
- https://codereview.qt-project.org/#/c/70783/
- https://bugreports.qt.io/browse/QTBUG-15602
- https://blog.qt.io/blog/2014/03/19/qt-weekly-2-localizing-qt-quick-apps/
- https://wiki.qt.io/How_to_do_dynamic_translation_in_QML

Thank you!

You can reach the author at
vmoolle@ics.com
(the demo source is at https://github.com/vmoolle/qtsummit2015demo)