Model Models: Tools for Making Better Behaved Models

Qt World Summit, 2019

Presented by André Somers

Qt World Summit 2019 Berlin

Introduction

Model types

I guesstimate that:

- ≈ 90% of models are lists
  - Perhaps with different aspects of the items in different columns
  - Includes almost all QML-consumed models
- 9% are trees
- 0.9% are real spread-sheet like tables
- 0.1% really are hierarchical monsters

QAIM API requirements

QAbstractItemModel requires very detailed change notification:

- Before and after
  - For inserts, removals, reset
- On changed

You often don't have all that from your data source... So how do you deal with that?

Real live "solutions"

Observed:

- Simply doing a full model reset
- Partial solutions handling insert or removals, but doing emitting a blanket dataChanged on the whole model
- Messy, hand-crafted in-place code that still cuts a few corners...
- Lots of duplicating the above for code with multiple updatable models...

Demo: talks/modelModels/ex-basic-example

Ideal:

- Simple oneliner that can be re-used...

Towards a solution

- Existing (stl) algorithms are not suitable:
  - No 'notifications'
- So define our own
  - Avoiding being dependent on container type or data type → templates!
- Basic assumptions:
  - List-like models
  - The model keeps a copy of the data internally in a random-access container
  - The updated data is set as a single block
  - Both are sorted (or can be sorted) in the same way on some non-changeable key

- Inputs:
  - `lessThan`
  - `hasChanged`
- Walk through input and exiting data at the same time
  - Using `lessThan` to see which one(s) to step
- Event-callbacks for needed inserts, removals, data updates
  - (and for equality too)
  - Event-callbacks handle actual update and model-signaling


```
1  //precondition: src and target are both ordered with respect to lessThan
2  template<ForwardIt FwdIt, Container TargetCollection,
3           BinaryPredicate LessThan, BinaryPredicate HasChanged,
4           EventHandler OnChanged, EventHandler OnInsert, EventHandler OnRemove, EventHandler
5  void updateCollection(const FwdIt srcBegin, const FwdIt srcEnd, TargetCollection& target,
6                        LessThan lessThan, HasChanged itemHasChanged,
7                        OnChanged onChanged, OnInsert onInsert, OnRemove onRemove, OnEqual onE
8  {
9      auto srcIt = srcBegin;
10     auto targetIt = std::begin(target);
11
12     while (srcIt != srcEnd) {
13         if (targetIt == std::end(target)) {
14             //insert: src has still items left while target has no more items
15              ~~~ }
16         } else if (lessThan(*srcIt, *targetIt)) {
17             //insert: src has one or more items that need to be inserted into target
18             auto srcInsertEnd = std::next(srcIt);
19             while (srcInsertEnd != srcEnd && lessThan(*srcInsertEnd, *targetIt)) {
20                 srcInsertEnd++;
21             }
22             targetIt = onInsert(srcIt, srcInsertEnd, targetIt);
23             //targetIt now points to the item after the item(s) just inserted
24             srcIt = srcInsertEnd;
25         } else if (lessThan(*targetIt, *srcIt)) {
26             //removal: target has items that are not in src (any more), so remove them
27              ~~~ }
28         } else {
29             //same item, check for changes
30              ~~~ }
31         }
32     }
```

Can we do better?

Much improved:

- Separate, readable blocks for comparing, detecting changes, inserting, removing and updating
- No loops (well, almost)
- Still a lot of code

Integrate in QAbstractModel

UpdateableModel template (simplified)

```
1  //actual class to inherit from
2  template<QAIM BaseModel, typename DataType>
3  class UpdateableModel: public BaseModel
4  {
5      //can't use Q_OBJECT on templates
6  protected: //methods
7      template<ForwardIt Iterator, Container DataContainer, BinaryPredicate LessThan>
8      Operations updateData(Iterator srcBegin, Iterator srcEnd, DataContainer& targetContainer
9                            LessThan lessThan, HasChangesFunction itemHasChanged)
10     {
11         Operations ops{0,0,0};
12
13         //events
14         auto onChanged = [this, &targetContainer, &ops](Iterator lhs, typename DataContainer
15
16         auto onInsert = [this, &targetContainer, &ops](Iterator lhsBegin, Iterator lhsEnd, t
17
18         auto onRemove = [this, &targetContainer, &ops](typename DataContainer::iterator rhsB
19
20         auto onEqual = [this](Iterator /*lhs*/, typename DataContainer::iterator /*rhs*/) {
21             flushCachedChanges();
22         };
23
24         updateCollection(srcBegin, srcEnd, targetContainer,
25                          lessThan, itemHasChanged,
26                          onChanged, onInsert, onRemove, onEqual);
27         flushCachedChanges();
28
29         return ops;
30     }
31
32     template<ForwardIt Iterator, Container Data>
```


UpdateableModel usage

- Subclass `UpdatableModel<QTableModel>` or `UpdatableModel<QListModel>`
- Implement `data()`, `flags`, `headerData`, etc. as you normally would
  - Taking care to keep these fast, of course...
- Add a setData that takes a new collection with an updated data set
- Provide a `lessThan`
  - As something you can put in a `std::function`
  - By reimplementing `UpdateableModel::lessThan`, or
  - By providing an `operator<` for your DataType
- Provide a hasChanges
  - Must return a data structure that contains which columns and which roles have changed
- Call `updateData()` from your `setData` and be done.

Resulting model update code

```
1  void ProcessModel::updateData(const ProcessList &processes)
2  {
3      const auto lessThan = [](const ProcessInfo& lhs, const ProcessInfo& rhs){return lhs.pid
4      const auto hasChanges = [](const ProcessInfo& lhs, const ProcessInfo& rhs){
5          DataChanges changes;
6          if (lhs.mem_res != rhs.mem_res) {
7              changes.changedRoles += RoleNames::MemRes;
8          }
9          if (lhs.mem_virt != rhs.mem_virt) {
10             changes.changedRoles += RoleNames::MemVirt;
11         }
12         if (!changes.changedRoles.isEmpty()) {
13             changes.changedColumns.append(0);
14         }
15         return changes;
16     };
17
18     auto changes = Base::updateData(processes.cbegin(), processes.cend(), m_processes, lessT
19 }
```

Demo: talks/modelModels/ex-proper-update

Features

- Can take any forward-iteratable data structure for source data
- Can deal with any storage container that:
  - Is forward-iteratable
  - Supplies insert that takes an iterator range, a number of values (and a copy) or a single value. SFINAE is used to select most efficient option.
- Tries to emit as few signals as possible:
  - Inserts and removals in blocks if possible
  - `dataChanged` in blocks of columns
    - Subject to policy on how to merge rows
- Returns number of operations performed (inserts, removals & updates)

Limitations

- Only supports list-like models, no trees
  - Mostly because there are no standard data structures for trees
- Only one set of changed roles per row
- Data needs to be sorted by key

Sorting

Sorting

QAIM limitations

QAIM does:

- Filtering
- Sorting

But how does it deal with changes

- in sort criteria (column or ascending/descending order)?
- in data affecting the sort order?

It uses the `layoutChanged` signal.

- No animations in QML.

Demo: talks/modelModels/ex-sorting-qsfpm

SortProxyModel

`SortProxyModel` reports changes in the order using row moves:

- for data changes
- for sort criteria changes
- drop-in replacement for `QSortFilterProxyModel`
  - for the sorting-part of its API

Demo: talks/modelModels/ex-sorting-sortproxymodel

Conclusions

Conclusions

Conclusions

- We can solve the issue of changing models in a generic way
  - Just not quite as a one-liner
  - ... but close enough.
- QSortFilterProxyModel does not provide moves
  - We can implement our own though

Source code is available under a liberal license as part of the KDToolBox github repository.

Questions?

Thank you for your time!

Contact us:

- **http://www.kdab.com**
- **KDToolbox: http://www.github.com/kdab/kdtoolbox/**
- **info@kdab.com**
- **training@kdab.com**
- **andre.somers@kdab.com**