ATF Qt Comparison

Public.

Revision History

The document compares the amount of work needed to create GUI game tools using ATF and Qt.

ATF Version	Revision Date	Author(s)	Changes
3.6	Dec 2013	Alan Beckus, Gary Staas	New document

Table of Contents

1 ATF Qt Comparison

Summary

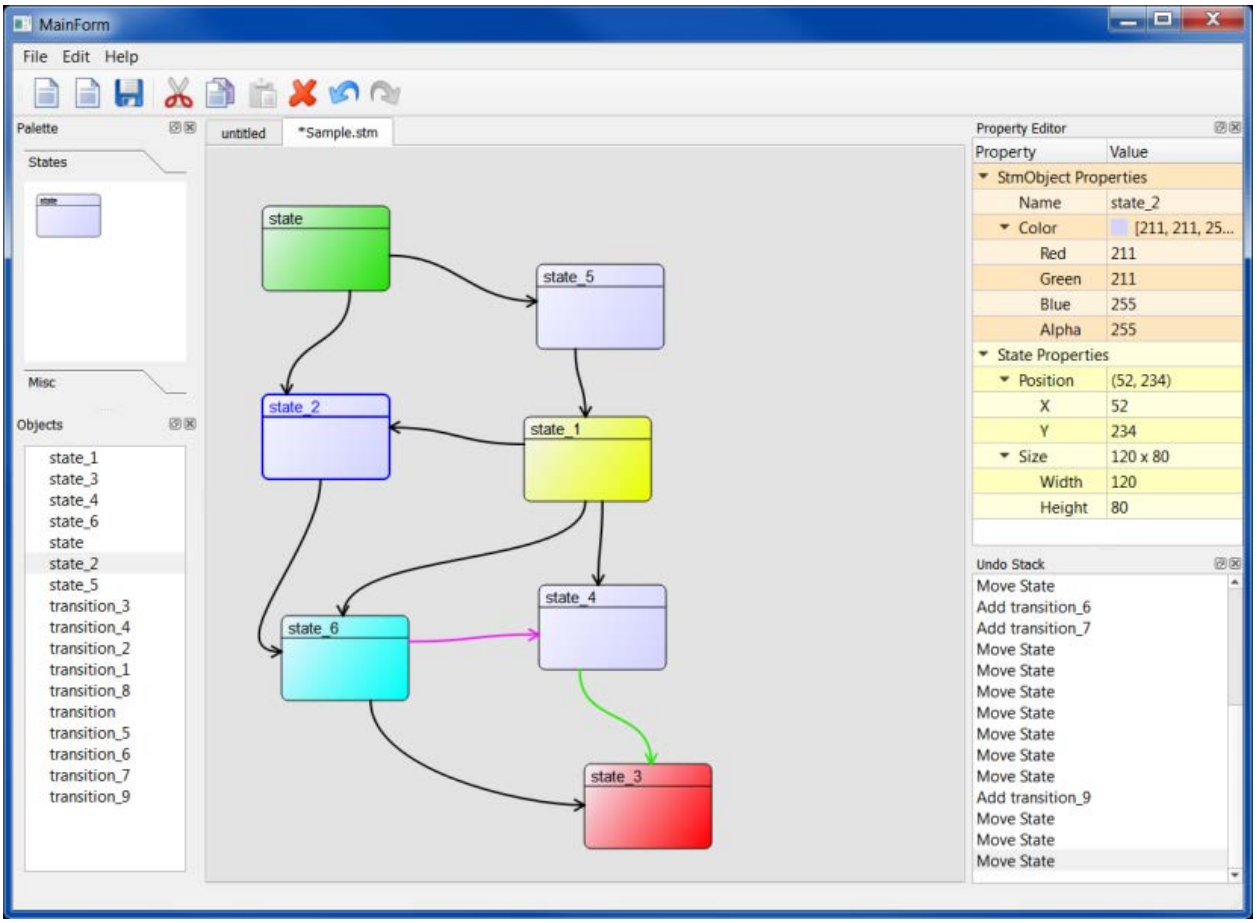
Read on for more details, but here is the conclusion: Using ATF to create game tools is a better choice than using Qt.

Example Application

To demonstrate the challenges of developing a game tool, ATF's Statechart Editor sample application is recreated in Qt/C++ as the `StmEditor` example application.

This example application uses one Qt component that requires a commercial license.

This screenshot shows the `StmEditor` example application developed in Qt/C++:



In order have a reasonably fleshed out `StmEditor` sample, prototypes of various framework services had to be implemented.

Framework Services

This section discusses, in no particular order, framework services already available in ATF that needed to be partially re-implemented or worked around to create this example application.

Document Related Services

Qt doesn't have a document model, so there are no document related services.

For this simple application, a `QTabWidget` is used to store all the open views. Each view owns one `StmDocument`.

Transaction Framework

A transaction framework monitors data for changes, and auto-creates and pushes `UndoCommand`s onto a history stack.

Implementing a command history (undo/redo) was particularly painful without the support of a transaction framework and observable data mode. More specifically, it was necessary to implement a new `UndoCommand` per object, per property. For real-life applications that have hundreds of object types, each with tens of properties, implementing all these commands causes big problems:

(1) Having to develop and maintain a large number of child classes that implement an `UndoCommand` base class is a significant effort.

(2) Even more troubling is direct coupling between call sites and each type of `UndoCommand`.

To avoid having to create numerous `UndoCommand`s, a robust transaction framework is required.

Per Document Selection Context

Qt doesn't allow deriving template classes from `QObject`. Consequently, a developer needs to implement something similar to a signal/slot to provide application wide notification. Since C++ does have an event system, there are a few implementations. For an example, see Fast C++ Delegate.

For `StmEditor`, a simple hard-coded multicast delegate was created using a class template, a functor, and a member function pointer. This quick and dirty solution was sufficient for this prototype sample application. For real applications, more time would need to be invested to create a more robust selection context.

For more details on `QObject` and the Meta-Object Compiler, see the "Limitations" section of Using the Meta-Object Compiler (moc).

Observable Data Model

Qt provides limited support for observable data via `QObject`, but it provides nothing near the capabilities of the ATF DOM (Document Object Model). The ATF DOM provides a powerful, flexible, and extensible mechanism for loading, storing, validating, and managing changes to large amounts of application or game data, independently of application code.

It is possible to retrofit signal/slots for this purpose, but they are cumbersome to use. It requires a prebuild step, and also there is no compiler warning or error to check the connection between signal and slot when using the Visual Studio add-in. And this does not provide the event propagation and adaptability that the ATF DOM does.

Property Editors and Related Interfaces

The base Qt doesn't come with property editors; these are only available for users who have a commercial license. A few basic property editors are included with Qt tools (Qt designer). However, the editors are not fully equipped to automatically detect object properties. Developers need to either implement a generic interface or an adapter, or programmatically populate the editor when the selection changes. This is another area where the developer can introduce a large number of couplings between the code that populates/updates properties and all the object types that need to be bound to property-editors—and this development is tedious, error-prone, and the resulting code hard to maintain.

Serialization

Serialization is required for loading and saving documents. Qt provides serialization for most of the primitive types, but there is no higher level or document level serialization. Users must create their own file format for each application.

The `StmEditor` sample has a simple text based format, which can be examined by looking at saved documents in Notepad.

In ATF, the `DomXmlWriter` and `DomXmlReader` classes take care of serialization.

Qt comes with an XML library, but it is similar to what .NET provides.

Control Host Service (Panel Host Service)

Qt has no service comparable to the ATF `ControlHostService` class, which provides a dock in which to hold controls, such as menus and toolbars.

Again, the developer needs to write code to handle all the panels. Or the developer can use the Qt Designer, but this is mostly for applications with a static UI.

Command Service

ATF's `CommandService` displays the currently available application commands to the user in menus and toolbars and provides keyboard shortcuts and other standard menu commands. There is no similar service in Qt. A user can create all the menu/toolbars in Qt Designer or programmatically create them during application initialization and bind a specific event to one or more slots.

Game Tool Specific Views and Controls

ATF provides many game tool specific classes, such as various kind of graph support, a curve editor, a timeline editor, and so on. For obvious reasons, Qt has none, because Qt is a generic UI toolkit, not made for specifically creating game related tools. Users need to implement any required UI widgets for each tool.

Qt comes with a generic Graphics View Framework and a scene based 2D graph capability. Users need to populate a scene with graphics elements, resulting in duplicate data: one in the user document and the other in the Qt `GraphicsScene`. For details, see Graphics View Framework.

Disregarding this above limitation, `GraphicsScene` seems a promising framework, but further investigation is required to determine if it is possible to use it to create a Circuit Editor type of view. Since it is scene based, it is uncertain whether it allows having wires from one scene node to another.

Model-View Architecture

Qt comes with a few Model-View based widgets (`QTreeView`, `QListView`, and `QTableView`). But their Model-View architecture is a bit over engineered, consequently harder to use—compared to ATF.

ATF's Model-View is based on a few well defined interfaces (`ITreeView`, `IListView`, and `IItemView`). A user can implement them in their document model.

Loading/Saving Application Data and User Preferences

Qt comes with `QSettings`, and this class can be used to store/load application settings and user preferences. However, it is only a half way solution, because it doesn't come with a Settings dialog to allow users to edit settings. For further detail, see "Detailed Description" in QSettings Class.

For `StmEditor`, no application settings were created. For more fleshed out applications, a generic setting service is recommend that automatically exposes some common settings and also allows users to define their own settings.

ATF's `SettingsService` manages user-editable settings (preferences) and the persistence of these application settings.

Conclusion

After going through this exercise, the author believes that using ATF to create game tools is a better choice than using Qt.

Even though Qt is a fairly robust framework that makes the unwieldy nature of C++ more accessible, ATF outshines Qt in almost every aspect of game tool development.

This difference is attributed to the following factors:

(1) ATF is built on C#/WinForms, so it inherits the ease of use and the rapid development of the .NET framework.

(2) ATF provides many services (building blocks) specifically designed for making game tools. It would be a big undertaking to implement similar services in Qt. It took the ATF team years to develop and refine these facilities.

Therefore, it will take years to do for Qt what ATF did for C#/WinForms.