Chapitre 4
Anatomie d'une application

1. Objectifs

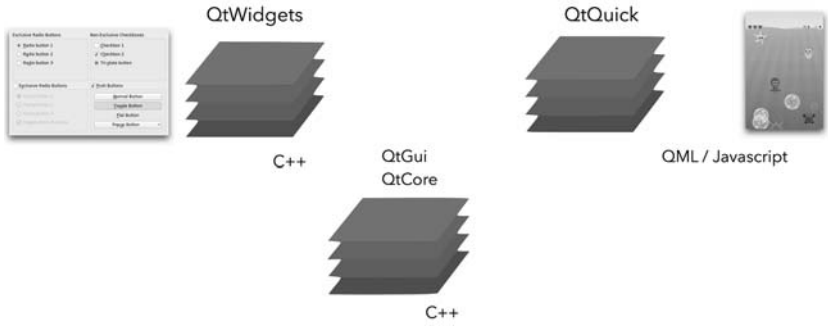
Nous l'avons dit précédemment, Qt est une bibliothèque. Mais c'est aussi un *framework* et à ce titre il impose un certain formalisme dans la conception et dans la programmation de vos classes. Ce formalisme suppose un certain travail humain dans la rédaction du code source, mais il induit aussi un certain travail automatique réalisé par les outils tiers livrés avec Qt.

Le SDK de Qt intègre les EDI Qt Design Studio qui permet de créer des interfaces graphiques Qt Quick, et Qt Creator. Ce dernier comprend un éditeur de code source, un outil de conception des interfaces graphiques Widgets, un débogueur et un outil d'analyse des applications (*profiler*). Il permet de s'affranchir de la ligne de commande pour la préparation de l'environnement de développement et pour la compilation du projet, mais ne remplace pas ces outils qui sont donc toujours utilisables pour les développeurs souhaitant travailler avec d'autres environnements de développement comme Xcode ou Visual Studio.

Dans ce chapitre nous montrerons comment est structuré un projet Qt, comment il est paramétré et comment se déroule la compilation de celui-ci jusqu'à la production du fichier objet, exécutable ou bibliothèque.

2. Application Widget ou Quick ?

La différence entre une application Widget et une application Qt Quick se situe uniquement au niveau des API de l'interface graphique. Ces deux technologies sont uniquement dédiées aux interfaces graphiques et s'appuient intégralement sur le modèle objet et les API du cœur de Qt, ainsi que sur tous les principes fondamentaux que nous avons précédemment exposés.



Dans le diagramme ci-dessus, vous pouvez constater que les usages de Qt Widgets et de Qt Quick sont très différents, Qt Quick étant beaucoup plus orienté graphisme que Widget. Les technologies mises en œuvre par Qt Widgets et Qt Quick sont également très différentes puisque dans le premier cas, les interfaces graphiques sont codées en C++ et en dans l'autre elles le sont en QML.

Les primitives d'affichage et les API du cœur (les chaines de caractères, les signaux et les *slots*, etc.) sont communes aux deux technologies et sont fournies par les mêmes modules Qt (Qt Gui, Qt Core, etc).

2.1 Application Widget

Une application utilisant les API de Qt Widgets pour son interface graphique aura l'aspect « classique » d'une application en mode fenêtré. De plus, les API de Qt sont faites pour que l'intégration au système d'exploitation soit la plus forte possible, vos fenêtres auront donc le même aspect que celles des autres applications. Ceci n'est pas toujours le cas avec d'autres frameworks comme Swing pour Java.

Une application développée avec Qt Widgets est purement C++. Aucun autre langage de programmation n'est utilisé. L'application fait un usage intensif des événements et de la programmation asynchrone, sa conception est très exigeante, c'est également le cas de sa programmation. Une application Qt Widgets performante nécessite un niveau avancé en programmation.

Il faut considérer l'utilisation de Qt Widgets pour tous types d'applications fenêtrées comme les applications métier, contenant des tableaux, des listes complexes, une interface graphique riche.

Une application Qt Widgets sera exécutée presque exclusivement sur des plateformes du type PC dotées d'un ou plusieurs écrans de grande taille.

2.2 Application Qt Quick

Une application utilisant les API de Qt Quick pour son interface graphique aura un aspect beaucoup plus moderne que son équivalent Qt Widgets. Cependant, elle aura moins l'aspect d'une application métier classique car ses polices sont plus petites, plus lissées, les composants actifs sont plus imposants. Finalement, tout donnera l'impression que l'application n'a pas été faite pour être exécutée sur un PC.

Et pour cause ! Qt Quick est surtout fait pour exécuter des applications sur les plateformes autres que PC : smartphones, tablettes, embarquées, équipées d'écrans tactiles principalement.

Une application développée avec Qt Quick conserve la plupart du temps tout son code métier en C++, mais toute la partie graphique est développée dans les langages QML et JavaScript. La conception d'une application Qt Quick est relativement simple, pour la partie graphique, la partie métier si elle est en C++, conserve des exigences de conception et de programmation identiques à Qt Widgets. Cette technologie est globalement plus accessible au développeur débutant.

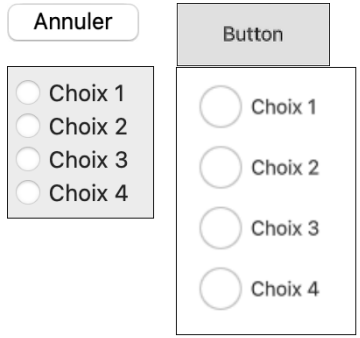
Il faut privilégier Qt Quick pour tous types d'applications multimédia, jeux et applications métier embarquées dans des équipements comme des tableaux de bord de voiture, des appareils médicaux, etc. Il est également très facile de développer de jolis tableaux de bord de supervision avec les API de Qt Quick.

Enfin, l'accès a été mis très fortement sur la 3D avec la dernière version de Qt, la version 6, avec un moteur 3D et une nouvelle abstraction du rendu 3D qui rend les performances nettement supérieures aux versions précédentes de Qt.

2.3 Synthèse

Une application Qt Widgets sera difficilement utilisable sur un petit écran ou sur un écran tactile. En effet, les composants graphiques de Qt Widgets sont prévus pour être manipulés avec une souris.

Comparons quelques composants Qt Widgets et leur équivalent Qt Quick :



Comme vous pouvez le constater, les composants Qt Quick sont sensiblement plus grands que leur équivalent Qt Widget. La différence est flagrante avec les boutons radio dont la taille du libellé est inférieure au bouton dans le cas de Qt Quick. La taille du libellé est également plus petite sous Qt Quick que sous Qt Widgets. Ce sont donc les éléments actifs qui occupent la place la plus grande.

Tout cela s'explique par la différence de précision entre un pointeur de souris, qui est précis au pixel sur un écran de PC, là où sur un écran tactile, la précision du doigt est à plusieurs dizaines de pixels près.

Le choix de l'une ou l'autre technologie devrait donc être fait au regard du type d'application, et surtout du type de plateforme d'exécution et des interfaces homme-machine (souris ou tactile).

3. Exemples d'architectures classiques

Dans cette section, nous étudierons quelques cas classiques d'applications et les choix qui peuvent être faits vis-à-vis des API.

3.1 Cartographie GPS

Les API de Qt Quick sont si puissantes aujourd'hui qu'il est possible de développer une cartographie GPS en quelques lignes de code :

```
import QtQuick 2.0
import QtQuick.Window 2.14
import QtLocation 5.6
import QtPositioning 5.6

Window {
 visible: true

 Plugin {
   id: mapPlugin
   name: "osm"

 }

 Map {
   anchors.fill: parent
   plugin: mapPlugin
   center: QtPositioning.coordinate(59.91, 10.75) // Oslo
   zoomLevel: 14
 }
}
```

Remarque

Ce code est extrait de la documentation officielle du Type QML Map. Cette documentation est accessible à l'adresse https://doc.qt.io/qt-5/qml-qtlocation-map.html. Veuillez noter que dans la version 6 de Qt, cette fonctionnalité est pour l'instant absente, elle sera portée dans une prochaine version. Si vous souhaitez utiliser le module de cartographie de Qt, vous devrez utiliser la version 5 de Qt.

L'exécution de ce code affichera la fenêtre suivante :

Il a suffit d'une vingtaine de lignes de code pour produire ce résultat. C'est toute la force de Qt Quick : il propose des API très riches.

Ceci étant dit, afficher une carte, c'est bien, mais cela ne suffit pas pour faire un véritable outil de cartographie. Il faudra afficher le point courant, proposer de saisir une adresse, récupérer les coordonnées GPS, enregistrer les dernières destinations, partager sa destination avec quelqu'un d'autres, et encore beaucoup de cas d'usage qui nécessiteront d'ajouter beaucoup de code, et de se retrouver assez rapidement confronté aux limites de Qt Quick.

Schématiquement, une application de ce type va s'appuyer fortement sur des composants graphiques riches et il ne sera pas nécessaire d'intervenir sur l'interface graphique depuis le code C++. Toute la gestion de l'interface avec l'utilisateur sera réalisée avec Qt Quick.

En revanche, tous les cas d'usage qui ne sont pas liés à l'interface graphique seront réalisés en C++, par exemple l'envoi d'un message, le stockage des données de l'application et de l'utilisateur sera géré par des classes C++.

Voici une synthèse des cas d'usage et des API utilisées :

Cas d'usage	Type	Modules	Langage
Affichage de la carte	Vue	Qt Quick, Qt Location	QML
Affichage du point GPS courant	Vue	Qt Quick	QML
Lecture du GPS	Contrôleur	Qt Positioning	QML
Saisie d'une adresse postale	Vue	Qt Quick	QML
Envoi de la destination par e-mail	Helper	Qt Network	C++
Stockage des destinations et routes	DAO	Qt Sql	C++
Stockage des données d'utilisation	DAO	Qt Core *	C++

* Pour des cas simples, il peut être utile d'utiliser l'API Qt Quick LocalStorage pour enregistrer des données d'utilisation ou des réglages.