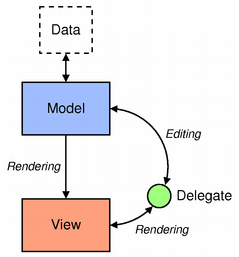
**Model/View Konzept**

In der Praxis werden viele User Interfaces mit dem MVC Pattern realisiert. Dieses Pattern besteht aus der View, dem Controller und dem Model. In diesem Zusammenhang wird eine strikte Trennung der einzelnen Schichten angestrebt. Das Ziel dieses Ansatzes, liegt in dem Austausch der View, ohne eine Anpassung der internen Datenstruktur durchführen zu müssen. Eine weitere Variante dieses Konzeptes ist das Model/View Konzept. Hierbei wird die View mit dem Controller kombiniert. Die Aufgabe der View es ist mit Hilfe des Models dem Benutzer die Informationen anzuzeigen. Der Controller reagiert lediglich auf Interaktion des Benutzers mit der View. Zusätzlich kann bei einer Model/View Architektur das Konzept eines „delegates“ eingeführt werden. Dieser besitzt die Aufgabe, die einzelnen Datenelemente des Models benutzerspezifisch anzuzeigen oder auf bestimmte Veränderungen des Datenbestandes von Seiten des Benutzers auf der View zu reagieren. Das Zusammenspiel der einzelnen Komponenten wird in Abbildung 1 nochmals graphisch veranschaulicht.

In Abbildung 1 wird deutlich, dass eine Trennung zwischen der Speicherung und der Darstellung der Daten besteht. Die Aufgabe des Models besteht darin, der View und dem Delegate minimalistische Schnittstellen für die Kommunikation bereitzustellen. Die Kommunikation zwischen den einzelnen Komponenten wird in QT mit Hilfe des Signal und Slot Konzeptes realisiert. Findet eine Änderung am Datenbestand des Models statt, wird ein Signal an die View und den entsprechenden Delegate geschickt. Diese rufen die entsprechenden Slots auf und aktualisieren die View. Im umgekehrten Fall, wenn der Benutzer die Daten via View verändert, schickt diese ein Signal an das Model und den Delegate. Ein weiteres Hilfsmittel der View ist der Modelindex. Dieser Index wird verwendet, um die einzelnen Informationen aus dem Datenbestand zu lesen.

Ein Modelindex ist eine Referenz auf einen einzelnen Datensatz des Models. Durch die Zuhilfenahme eines Delegates, kann dieser benutzerspezifisch auf der View dargestellt werden. Diesbezüglich muss erwähnt werden, dass es mehrere Möglichkeiten für die Erstellung der View unter QT existieren. Eine Möglichkeit ist die Erstellung mittels QT Widgets. Diese können durch Qt bereitgestellte Klassen erzeugt werden. Hierbei ist eine eindeutige Trennung zwischen View und Model schwer möglich. Eine andere Variante ist die Erstellung mittels QML. QML ist eine Art Beschreibungssprache für Benutzeroberflächen. Hierbei muss lediglich eine Beschreibung der GUI angegeben werden. Diese ist so allgemein gehalten, dass Designer oder UI-Entwickler ohne irgendwelche Vorkenntnisse einer Programmiersprache eine View anfertigen können. Der Vorteil hiervon liegt in der klaren Trennung der einzelnen Aufgaben. Ein Designer kann sich ausschließlich um das UI kümmern und ein Software-Entwickler um das dazugehörige Model und den Controller. Diesbezüglich existiert eine klare Trennung zwischen dem Model und der View.

In Qt sind alle Model Klassen von der Abstrakten Basisklasse QAbstractItemModel abgeleitet. Diese Basisklasse bietet eine Vielzahl an Schnittstellen für die Kommunikationen mit der View an. Um auf eine gegeben Datenstruktur besser reagieren zu können, bietet QT 3 besondere Model-Typen an. Diese Modeltypen sind: QListModel, QTableModel und QTreeModel. Mit Hilfe dieser 3 Modeltypen, kann eine Vielzahl der Anwendungsbereiche abgedeckt werden. In diesem Projekt ist ausschließlich ein QAbstractListModel verwendet worden. Zur Umsetzung des Model View/Konzeptes wurde für die Bereitstellung der UI die Beschreibungssprache QML verwendet. Für die Implementierung des Models wurde die bereits von Qt bereitgestellte Klasse QAbstractListModel verwendet. Hierbei ist eine neue Unterklasse von QAbstractListModel erzeugt und die entsprechenden Methoden für die Kommunikation mit der View neu implementiert worden. Als Datenstruktur wurde eine QList mit entsprechenden Datenobjekten gewählt. Die Datenobjekte besitzen die Aufgabe, die einzelnen Messwerte einer Messung zu kapseln. Eine Vielzahl der QML Elemente wie beispielsweise eine ListView oder TableView bieten standartmäßig eine Property „model“ an, welche die Verknüpfung mit dem entsprechenden Model realisiert. Des Weiteren kann mittels QML ein delegate Objekt erzeugt werden. Mit dessen Hilfe, können beispielsweise die Einträge in einer ListView bestmöglich auf die Wünsche des End-Benutzers angepasst werden.

Der Vorteil des QT Frameworks ist die automatische Anpassung der GUI an eine Änderung des internen Datenbestandes. Der Entwickler muss lediglich dem Model mitteilen, wann eine Änderung am Datenbestand des Models durchgeführt wurde. Infolgedessen übernimmt das Framework die komplette Aktualisierung der GUI. Der umgekehrte Fall, dass der Benutzer die Daten mittels GUI verändern kann, ist im Projektverlauf nicht implementiert worden.

Für die bestmögliche Darstellung der gesammelten Daten, mussten mehrere Diagramme erstellt werden. Hierbei bestand die Möglichkeit, alle Diagramme über die von QT bereitgestellten Klassen zu erzeugen, oder ein bereits vorhandenes auf QT basierendes Modul namens QCustomPlot zu verwenden. Dieses Modul kapselt die von QT bereitgestellten „Paint“ Klassen und gibt dem Benutzer eine Vielzahl an bereits vorimplementierten Diagramm-Typen. Ursprünglich wurde diese Third-Party Modul für den Einsatz mit QT Widgets konzipiert. Diesbezüglich musste eine Portierung in QML durchgeführt werden. Folglich konnten einige Funktionalitäten wie beispielsweise das Zoomen nicht in QML überführt werden. Die Portierung ist mittels der QCustomPlot Support Seite durchgeführt worden.

**Quellen:**

<http://qt-project.org/doc/qt-4.8/modelview.html>

<http://qt-project.org/doc/qt-4.8/model-view-programming.html>

<http://sysmagazine.com/posts/181712/>

<http://doc.qt.io/qt-5/qtqml-cppintegration-interactqmlfromcpp.html>

<http://qt-project.org/doc/qt-4.8/qmlevents.html>

<http://qt-project.org/doc/qt-4.8/qtbinding.html#exchanging-data-between-qml-and-c>

<http://www.qcustomplot.com/index.php/support/forum/172>

<http://www.qcustomplot.com/index.php/tutorials/settingup>

The Book of QT 4.0 Daniel Molkentin