Design- und Implementationsdokumentation für WeatherInfo

SE3 Team
Tobias Koenig

Juni 2009

Inhaltsverzeichnis

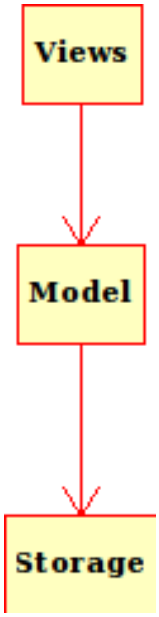
1 Einleitung

Im Anschluss an die Analysephase, deren Ergebnisse im Dokument analyse.pdf zusammengefasst sind, wurden in der Designphase der grobe und feine Aufbau des Beispielprogramms WeatherInfo entwickelt. Da das Model-View-Controller Konzept ein integraler Bestandteil der Applikation darstellt, wurde der grobe Aufbau dadurch implizit schon festgelegt. Der Feinentwurf definierte die einzelnen Komponenten des Programms, welche eine nahezu 1:1 Repräsentation im Quellcode haben, und die Kommunikation zwischen ihnen.

2 Der Grobwurf

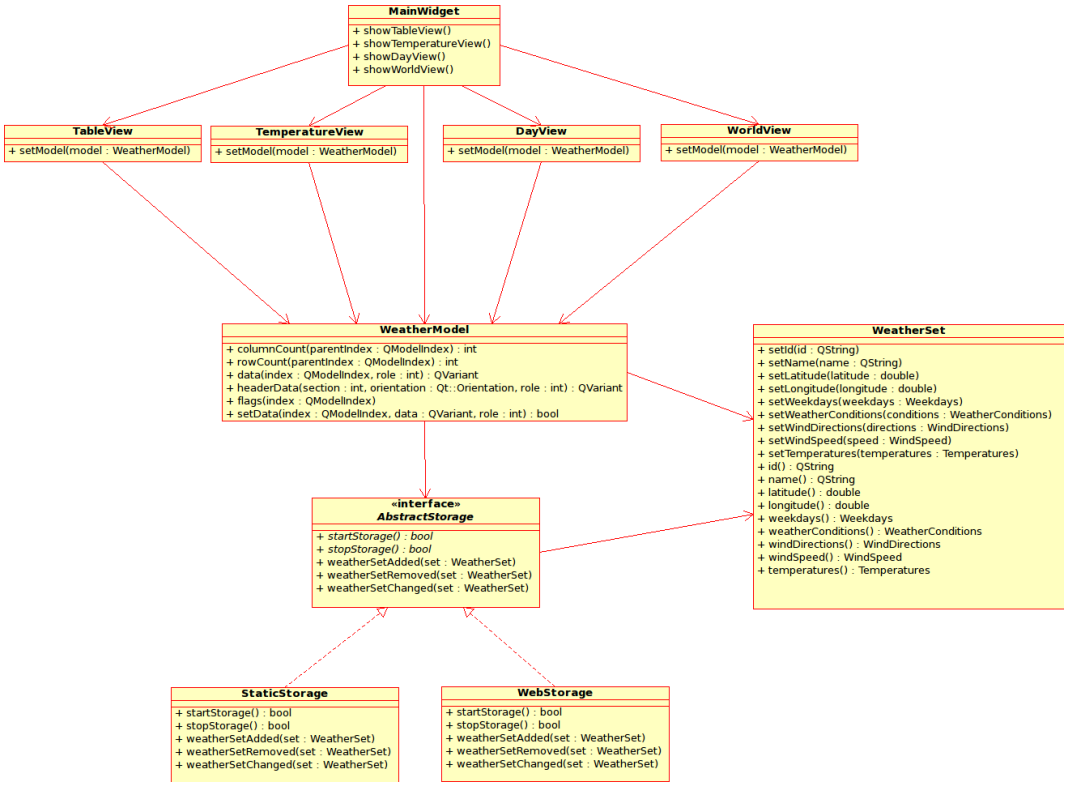
Das Programm lässt sich grob in drei Module unterteilen:

- **Views** Das Views-Modul enthält alle visuellen Komponenten des Programms und definiert welche Interaktion zwischen den Komponenten stattfinden kann.
- **Model** Das Model-Modul enthält das Daten-Model und die Datenstrukturen, welche zur Kommunikation zwischen dem Model-Modul und Storage-Modul benötigt werden.
- **Storage** Das Storage-Modul enthält die Komponenten welche zum Laden der Daten vom WebService benötigt werden.



3 Der Feinentwurf

Im Feinentwurf wurde der Grobentwurf dahingehend überarbeitet, dass das Views-Modul jetzt in die einzelnen View-Komponenten zerlegt wurde und die Beziehungen zwischen ihnen verdeutlicht. So stellt die *MainWidget* Komponente das Hauptfenster dar, aus dem heraus die anderen Views aktiviert werden können. Sowohl das *MainWidget* als auch die anderen Views haben eine Assoziation zu dem *WeatherModel*, welches den Model Aspekt des MVC in diesem Programm wiederspiegelt. Das *WeatherModel* seinerseits hat eine Assoziation zum *AbstractStorage* bzw. einem seiner beiden konkreten Implementierungen *StaticStorage* oder *WebStorage*. Diese beiden Komponenten sind dafür verantwortlich die Wetterdaten aus einem lokalen Speicher bzw. aus dem Internet zu Laden und an das *WeatherModel* zu übergeben. Als Containerstruktur für diese Daten wird *WeatherSet* verwendet.



Im Diagramm ist zu sehen, dass eine Assoziation vom Storage zum Model, bzw. vom Model zu den Views fehlt. Das ist dem Umstand geschuldet, dass der Datenaustausch bzw. die Benachrichtigung über Datenänderungen nicht über Funktionsaufrufe realisiert

wird, und damit der Empfänger der Daten und dessen Schnittstelle bekannt sein muss, sondern über den Signal/Slot-Mechanismus des Qt-Frameworks, welcher eine Bindung der Komponenten zur Laufzeit gestattet. Dadurch wird eine zusätzliche Entkopplung zwischen den Komponenten erreicht und im Gegensatz zum original MVC Paradigma muss das Model keine Informationen über den View besitzen, der View entscheidet gegen welches Model er sich bindet und auf welche Signale er reagieren möchte.

4 Die Implementation

Die Entwicklung des Programms wurde basierend auf dem Qt-Framework durchgeführt, welches es ermöglicht, plattformübergreifende Software in C++ zu implementieren. Neben Funktionen zum Anzeigen von Fenstern und Malen von Graphiken beinhaltet es ein Model-View-Framework, was den modularen Entwurf und Implementierung von Programmen zum Anzeigen von Daten aus verschiedenen Datenquellen in unterschiedlichen Ansichten erheblich vereinfacht. Das *MainWidget* und die verschiedenen Views wurden von *QWidget* abgeleitet und haben damit eine visuelle Repräsentation auf dem Bildschirm. *WeatherModel* ist eine Subklasse von *QAbstractTableModel*, welches wiederum von *QAbstractItemModel* ist und damit das Basis-Interface des Qt Model-View-Frameworks zur Verfügung stellt. Eine konkrete Implementation von *QAbstractItemModel* kann man an verschiedenste Klassen des Qt-Frameworks übergeben, welche die Daten unterschiedlich darstellen. *QTableView* listet die Daten in einer Tabelle auf, wohingegen *QListView* nur die erste Spalte des Models in einer Liste darstellt. *QComboBox* wiederum bietet dem Benutzer an, die Daten einer definierten Spalte des Models aus einer Drop-Down Box auszuwählen. In WeatherInfo wurde von den vordefinierten Views nur *QTableView* für die Tabellenansicht der Wetterdaten verwendet.

Eine weitere verwendete Funktionalität des Qt-Frameworks sind die Proxy-Models. Diese Klassen, welche von *QAbstractProxyModel* erben, bekommen ein *QAbstractItemModel* als Source-Model und bieten selbst ebenfalls das Interface des *QAbstractItemModel* an. Sie können damit zwischen einer datenbasierten *QAbstractItemModel* Implementation und einem View geschoben werden und Filter-, Sortier- oder Strukturänderungsfunktionen übernehmen. WeatherInfo macht davon Gebrauch um die Anzeige der Städte in der Tabellenansicht zu filtern.