1 Data Binding

Eine Möglichkeit, Verknüpfungen von Daten zu schaffen, ist das Binden von Daten an Properties, auch Data Binding genannt. Properties sind Objekte die das Property Interface implementieren welches unter anderem aus dem Observable Interface besteht. Die Idee ist, dass sich andere Objekte, die selber das Property oder Observer Interface implementieren, an Properties registrieren und automatisch bei Änderungen informiert werden und entweder den Wert übernehmen oder mittels Converter umwandeln. Verwendet wird diese Technik um das UI mit Daten zu versorgen. Anstatt dies über Methoden im Codebehind zu realisieren, werden die Properties im UI mit den Properties vom Datenmodell oder Controller verbunden (Mehr dazu unter Entwurfsmuster). Beispielsweise wird die TextProperty eines Textfeldes an das Feld ownPoints gebunden:

```
ownPoints.textProperty().bind(viewModel.ownPoints());
```

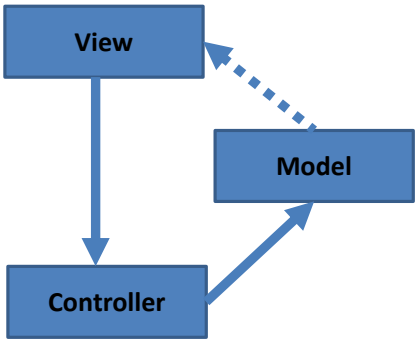
Außerdem wird der im Root-Element der FXML-Struktur angegebene Controller über die FXML-Struktur vererbt. Somit können auch alle darunterliegenden Elemente darauf zugreifen und müssen bei Verwendung nur noch den Pfad zu dem benötigten Feld oder der Methode angeben.

2 Entwurfsmuster

2.1 Model-View-Controller

Als Entwurfsmuster für Benutzeroberflächen ist das Model-View-Controller (MVC)-Entwurfsmuster sehr verbreitet. Es basiert auf drei Komponenten, welche die Präsentation von den Daten und der Interaktion trennen.

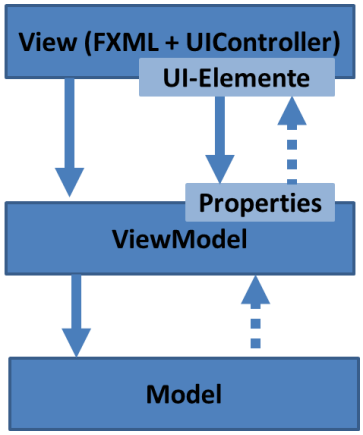
Die Komponente Model beinhaltet das Datenmodell und informiert die View, falls es Änderungen gibt. Die View reagiert auf Änderungen im Model und stellt die Informationen in der Benutzeroberfläche dar. Aktionen, die von der Benutzeroberfläche kommen nimmt der Controller entgegen und verändert beispielsweise das Model.



MVC ist allerdings nicht optimal für das zu erstellende Programm geeignet, da keine Schicht zwischen View und Model existiert. Somit ist eine Umsetzung nur nach diesem Entwurfsmuster schwierig, da die Benutzeroberfläche nicht direkt mit dem Datenmodell arbeiten, sondern ein extra UI-Datenmodell erstellt werden soll um das reine Datenmodell von der Aufbereitung für das UI, der Bereitstellung von Properties, zu trennen. Deshalb haben wir nach einer ersten Umsetzung nach MVC unsere Architektur auf das im Folgenden beschrieben Modell umgestellt.

2.2 MVVM

Als passende Alternative hat sich das Model-View-ViewModel (MVVM)-Entwurfsmuster angeboten, das eine bessere Trennung von UI-Design und UI-Logik ermöglicht.



Das Model und die View sind dabei mit denen des MVC-Entwurfsmusters vergleichbar. Wobei die View nur das ViewModel kennt und nicht das Model. Außerdem befindet sich im View keine Ablauflogik, diese wird vom ViewModel übernommen, das Properties bereitstellt, die von der View mittels Data Binding genutzt werden. Somit besitzt das ViewModel keine Kenntnis darüber, von welchen Views es referenziert wird. Es greift lediglich auf das Model zu und bereitet dessen Daten auf und reagiert auf Aktionen im UI. Damit erweitert es die Funktionalität des Controllers vom MVC-Entwurfsmuster um die Logik zum Aufbereiten der Daten für die View. Kritik wird am MVVM-Entwurfsmuster vor alle geübt, weil das ViewModel zu viele Aufgaben zu erledigen hat, da es einerseits die Interaktion mit dem Model übernimmt, also auch validierte Daten an das Model übergeben soll, und andererseits die Daten für das View aufbereiten soll. Deshalb wird vorgeschlagen, das ViewModel in zwei Schichten zu teilen, die diese beiden Aufgaben getrennt voneinander erledigen. Da sich der Funktionsumfang bei dem geplanten Programm in Grenzen hält, sollte dies nicht nötig sein und zu keinem Problem führen.