MVC

Das ***model-view-controller*** – Konzept (ein Entwurfsmuster: MVC-pattern) ist ursprünglich für Anwendungen der OO Sprache Smalltalk entwickelt worden. Inzwischen ist es ein Standard-Konzept der Softwareentwicklung für Anwendungen mit grafischen Oberflächen geworden.
Das Ziel ist, die Behandlung der Daten unabhängig von ihrer Darstellung zu machen. Damit werden Änderungen der Darstellung keinen Einfluss auf die Daten haben, was beispielsweise bei Wechsel zwischen verschiedenen Betriebssystemen wichtig ist, aber auch die Wiederverwendung einer Model-Komponente in einem anderen Kontext leichter macht. Das im Zusammenhang mit dem Prinzip der Kohäsion angesprochene Konzept der losen Kopplung wird hier umgesetzt.
Gamma[1]schreibt:
"Im MVC-Paradigma werden View und Model-Objekte durch den Aufbau eines Protokolls zur Benachrichtigung entkoppelt. Ein View-Objekt muss sicherstellen, dass seine Darstellung den Zustand des Model-Objektes wiedergibt. Das Model benachrichtigt die von ihm abhängigen Views, wenn sich seine Daten ändern."
Rappin/Dunn gehen im Buch ***wxPython in Action*** intensiv auf diese Probleme und das möglichst ernsthafte Verfolgen des MVC-Musters ein. An deren Beschreibung orientiere ich mich im folgenden Text[2]. Siehe aber auch Wikipedia[3].

Die Model[4] – Komponente

Die Model-Komponente beschäftigt sich mit der *Geschäftslogik*, das sind in der Regel Daten oder in Daten abgebildete Inhalte, also das, was der eigentliche Inhalt des Projektes ist. In unserem Moebel-Projekt sind das die ganzen Möbelklassen – von welcher Entwicklungssstufe auch immer.
Die Beziehung zu den anderen Komponenten ist geprägt vom *Beobachter – Entwurfs-muster*. Wenn es Änderungen im Modell gibt, die für den Controller oder eine View-Komponente wichtig sind, müssen diese informiert werden. Der Begriff Beobachter ist nicht ganz präzise. Sie beobachten den Zustand vom Model nämlich nicht aktiv. Statt dessen werden sie über dafür bestimmte Methoden angesprochen [sie erhalten eine message]. Im Wikipedia-Text heißt es daher *Das Modell ist das zu beobachtende Subjekt, auch Publisher, also „Veröffentlicher", genannt. Aktiv ist also die Model-Komponente.*

Die View[5] – Komponente

Das wird in vielen Fällen nicht eine Komponente sein. Auch in unserem Beispiel mit der Raumplaner-Grafik und der Gui zur Steuerung benutzen wir zwei View-Komponenten. In unserem Fall geschieht die Darstellung der benötigten Daten aus dem Modell bisher allein durch ihr Bild, allerdings wäre auch eine tabellenartige Darstellung der gespeicherten Daten sinnvoll. Die Gui ist in der bisherigen Version allein für die Entgegennahme von Benutzerinteraktionen zuständig.
Es kommt hinzu, dass es in vielen Fällen eine geschachtelte Struktur solcher grafischer Elemente gibt, z.B. ein pop-up-Fenster, das abhängig ist von einem anderen, geöffnet wird und auf eine Eingabe wartet. Auch hier tritt außerdem das Kompositum-Muster auf,

1 Gamma e.a. "Entwurfsmuster"; Addison-Wesley
2 Die Folge ist, dass einzelne Sätze schlichte Übersetzungen des Englischen Textes sind.
3 http://de.wikipedia.org/wiki/Model_View_Controller
4 Die Deutsche Schreibweise ist Modell.
5 Deutsch: Die Präsentation

da es Fensterelemente gibt, die für sich wieder View-Komponenten sind.

Die Controller[1]-Komponente

Bei Wikipedia heißt es: *Die Steuerung verwaltet eine oder mehrere Präsentationen, nimmt von ihnen Benutzeraktionen entgegen, wertet diese aus und agiert entsprechend. Zu jeder Präsentation existiert eine Steuerung. Es ist die Aufgabe der Steuerung, Daten zu manipulieren. Die Steuerung entscheidet aufgrund der Benutzeraktion in der Präsentation, welche Daten im Modell geändert werden müssen.*
Das sieht bei unserem einfachen Grafikfenster gar nicht so aus. Allerdings gibt es auch bei ihm eine Aktion, die an den Controller weiter gereicht werden muss, nämlich das Schließen des Fensters, worauf der Controller mit Beenden der Anwendung reagieren soll. Da das reine Grafikfenster keine weiteren Aktionen entgegennimmt, ist bis hierhin eine eigene Controller-Klasse verzichtbar. Wichtig wäre, eine saubere Trennung von Modell und View zu überprüfen.

Wegen der Gui den Controller entwickeln

Die Aktionen der Gui sollten an das Controller-Objekt weiter gereicht werden. In einfachen Anwendungen wird oft auf eine gesonderte Controller-Klasse verzichtet[2]. Es kommt hinzu, dass oft Funktionalität des Controllers in die einzelnen Elemente grafischer Benutzeroberflächen integriert sind. Im wxPython-Buch heißt es dazu [übersetzt]:
In vielen modernen UI toolkits sind die beiden Komponenten [View und Controller] mit einander verstrickt. Das liegt daran, dass Teile der Controller-Funktion selbst auf dem Bildschirm dargestellt werden müssen und dass Fensterelemente, die Daten darstellen, auch auf Ereignisse reagieren sollen.
Ein Beispiel ist ein Eingabefeld, dass ein Ereignis "*Text geändert*" erzeugt. *Diese Möglichkeit ist bei wxPython von vorn herein in jede Fenster-Komponente (erbt von wxWindow) eingebaut, da sie außerdem von wxEventHandler erbt.*
Im Buch wird ausführlich der Kontrollfluss beschrieben. Erreicht den Controller ein Ereignis, dann ordnet er diesem die Folgen für das Modell zu und ruft die notwendigen Methoden auf. Ändert das Model seinen Zustand, teilt es dem Controller mit, dass die Daten verändert worden sind, löst also eine Update-Aktion aus. Diese wird durch die entsprechenden Methoden an die View-Komponenten weiter gegeben, die nun die Darstellung ändern.
Zulässig ist nach dem MVC-Konzept auch, dass die Update-Aufforderungen direkt an die View-Komponenten weiter gegeben werden. Wir haben bei unserem (Anfangs-) Raumplaner-Projekt genau diese Situation: Die Moebel-Objekte senden an das Zeichenflaeche-Objekt die messages **`Loesche(...)`** und **`Zeichne(...)`**, das Zeichenflaeche-Objekt holt sich die notwendigen Daten von den Moebel-Objekten und aktualisiert die Darstellung.
Die interessante Modellierungsfrage ist: Muss das Zeichenflaeche-Objekt eigentlich die Objekte kennen?

1 Deutsch: Steuerung
2 Das Standard-Gui-Konzept, auf das man bei der Gui-Entwicklung mit dem Java-Editor von Gerhard Röhner trifft, arbeitet mit diesem Weg.

Eine Neumodellierung

Beim aktuellen (2023) neuesten Projekt ist das MVC-Konzept vollständig neu umgesetzt. Für das Modell relevante Veränderungen werden in dieser Variante grundsätzlich über den Controller angestoßen.

Ereignisse

Die laufende Anwendung muss von zwei (*je nach Version drei)* View-Komponenten ausgelöste Ereignisse verarbeiten. Diese beiden Komponenten sind:

Die Zeichenfäche

Hier sind es die Mausereignisse, die verarbeitet werden müssen. Beobachter ist der Controller, der über das Auftreten der Mausereignisse informiert wird.

- ***Mausklick*** (*OnLeftDown*) mit der linken Maustaste auf der Zeichenfläche wird zum Steuern der Auswahl von Möbelobjekten genutzt. Wird ein Möbelobjekt angeklickt, wird es ausgewählt, umgekehrt wird die Auswahl gelöscht, wenn in einen freien Bereich geklickt wird.
- ***Rechter Mausklick*** (*OnRightDown*) wird in der Version mit Konfigurationsdialog genutzt, um diesen für das angeklickte Möbelobjekt zu öffnen.
- ***Maus loslassen*** (*OnLeftUp* und *OnRightUp*) beenden diese Zustände.
- ***Maus ziehen*** und ***Maus bewegen*** (beide *OnMotion*) können mit der Abfrage ***Taste ist unten*** (event.LeftIsDown()) im Zusammenhang verarbeitet werden.

Die Gui

Wichtige zu verarbeitende Ereignisse stammen von der Gui.

- Im Menü kann unter Datei das ***Laden*** und ***Speichern*** von Daten angefordert werden.
- Weiterhin können im Menü die Möbelobjekte erzeugt werden.
- Ein weiteres Menü ermöglicht die Auswahl der Möbelobjekte und bietet die Möglichkeit, das ausgewählte Objekt zu verbergen und es erneut anzuzeigen.
- Bei der einfachen Oberfläche wird im Textfeld (*TextCtrl*) auf der Inhaltsfläche (*panel*) der Gui vom ausgewählten Möbelobjekt seine Position in der Liste und der Klassenname angezeigt. Mit Mausklick auf einen der Buttons kann ausgewählt bzw das ausgewählte Objekt jeweils um 30° gedreht werden.

Der Konfigurationsdialog

Der Konfigurationsdialog bietet die Möglichkeit alle Daten des ausgewählten Möbelobjekts zu verändern. Dabei werden Label (*StaticText*) und Textfelder verwendet und ein Grid-Layout eingesetzt.

Was macht der Controller?

Der Controller muss zu allen diesen Anforderungen passende Methoden bereit stellen. Wichtig ist dabei, dass jede Änderung des Zustands im Modell, den er ausgelöst hat, über eine Update-Methode an die **Beobachter Zeichenfläche und Gui** weiter gereicht werden.
Besonders ist an der letzten Modellierung mit einer Oberfläche, dass die Zeichenfläche sich die Daten zur Darstellung nicht direkt bei den Möbelobjekten holt.
Das ist grundsätzlich möglich und nach dem MVC auch zulässig. Die Zeichenfläche muss sich dann nur die Liste der Objekte vom Controller holen, bei den Objekten abfragen, ob sie sichtbar sind und für diese dann die Figur, und die Farbe (ggf die FuellFarbe) holen. Dazu nutzt sie bei der Variante die entsprechenden Methoden *GibSichtbar()*, *GibFigur()*, *GibFarbe()* und *GibFuellFarbe()* der Möbelklassen.

Neu

In der neuen Variante stellt der Controller der Zeichenfläche nicht die Objekte bereit, sondern nur die Figuren mit Farbe und Füllfarbe.
Eine Ursache für diese Neumodellierung war die Schwierigkeit bei einigen Möbelobjekten bei wxPython alle inneren Flächen wie gewünscht gleichmäßig zu färben[1]. Figuren können bei dieser Variante auch gegebenenfalls nur Teilfiguren des Möbelobjekts sein.

1 Sollen Teile der Möbel mit unterschiedlichen Farben in der Füllung und ggf auch des Rands dargestellt werden, muss man auf dieser Modellierung aufbauend noch einige weitere Änderungen umsetzen.

Anlage

Text aus Wikipedia:

Modell (model)

Das Modell enthält die darzustellenden Daten und gegebenenfalls (abhängig von der Implementierung des MVC-Patterns) auch die Geschäftslogik. Es ist von Präsentation und Steuerung unabhängig. Die Bekanntgabe von Änderungen an relevanten Daten im Modell geschieht nach dem Entwurfsmuster „Beobachter“. Das Modell ist das zu beobachtende Subjekt, auch Publisher, also „Veröffentlicher“, genannt.

Präsentation (view)

Die Präsentationsschicht ist für die Darstellung der benötigten Daten aus dem Modell und die Entgegennahme von Benutzerinteraktionen zuständig. Sie kennt sowohl ihre Steuerung als auch das Modell, dessen Daten sie präsentiert, ist aber nicht für die Weiterverarbeitung der vom Benutzer übergebenen Daten zuständig. Im Regelfall wird die Präsentation über Änderungen von Daten im Modell mithilfe des Entwurfsmusters „Beobachter“ unterrichtet und kann daraufhin die aktualisierten Daten abrufen. Die Präsentation verwendet oft das Entwurfsmuster „Kompositum“.

Steuerung (controller)

Die Steuerung verwaltet eine oder mehrere Präsentationen, nimmt von ihnen Benutzeraktionen entgegen, wertet diese aus und agiert entsprechend. Zu jeder Präsentation existiert eine Steuerung. Es ist die Aufgabe der Steuerung, Daten zu manipulieren. Die Steuerung entscheidet aufgrund der Benutzeraktion in der Präsentation, welche Daten im Modell geändert werden müssen. Sie enthält weiterhin Mechanismen, um die Benutzerinteraktionen der Präsentation einzuschränken. Die Steuerung kann in manchen Implementierungen ebenfalls zu einem Beobachter des Modells werden, um bei Änderungen der Daten den View direkt zu manipulieren.
Da das MVC-Muster in verschiedenen Programmiersprachen unterschiedlich realisiert werden muss, gibt es selbst keine genaue Definition über die Positionierung der Geschäftslogik innerhalb der MVC-Klassen. Diese kann je nach Anwendungsfall besser im Controller aufgehoben sein oder auch in das Modell verlagert werden. In der Praxis finden sich unterschiedliche MVC-Frameworks: Einige schreiben strikt vor, wohin die Geschäftslogik gehört, andere überlassen diese Entscheidung dem Softwareentwickler.

Nicht definierte Funktionalitäten

In ähnlicher Weise ist der Ort für die Validierung der Benutzereingaben nicht definiert. Einfache Formatvalidierungen können bereits im View realisiert werden. Validierungen, welche stärker die Geschäftslogik berücksichtigen müssen, werden eher im Model oder im Controller implementiert.
Auch für die Formatierung der Rohdaten und die Internationalisierung ist nicht definiert, wo diese erfolgen. Aus Gründen der Entwicklungseffizienz bietet es sich oft an, diese im Model zu integrieren, so dass man sich beim View auf die Erstellung von Widgets oder

Templates beschränken kann. Andererseits werden dadurch Aspekte der Darstellung in das Model verlagert, was zur Grundidee durchaus im Widerspruch steht. Als Variante bietet es sich daher auch an, hierfür eigenständige Funktionsbereiche vorzusehen, die man dann weder Model, View oder Controller zurechnen muss.

Widget-Bibliotheken für Desktop-Applikationen

Als Widgets werden die einzelnen Komponenten grafischer Oberflächen bezeichnet, wie Menüpunkte oder Editor-Komponenten. Widgets zeichnen sich dadurch aus, dass sie neben der Präsentation auch typische Merkmale des klassischen Controllers in einer Komponente vereinen, wie das Event-Handling. Einige Widgets, wie z. B. Auswahllisten, können sogar über ein eigenes internes Modell verfügen, wobei dieses dann mit dem eigentlichen Modell synchronisiert werden muss.

Obwohl die Widgets die feste Dreiteilung durchbrechen, spricht man trotzdem noch von einer Model-View-Controller-Architektur. Es kommen auch Komponenten wie Filter zur Sortierung oder Bestätigungsdialoge vor, die sich nicht eindeutig in die klassische Dreiteilung einordnen lassen.

Bei der Anwendung der Widget-Bibliotheken überlässt der Controller damit einen Teil der klassischen Controller-Funktion den Widgets und beschränkt sich auf die Steuerung des Models und gegebenenfalls anderer Komponenten des Views.