CMF 3.0 FlowView

Inhaltsverzeichnis:

Layout im Großen (Sammlung von Informationspartikel -> im View)

org.sociotech.cmf3.view.flow.FlowView < FXView < View < VisualGroup < VisualComponent < javafx.scene.Group

- Instanzvariable: javafx.scene.Scene scene (wird im Konstruktor erzeugt - mit width, height)
- Instanzvariable: org.sociotech.communitymashup.data.DataSet
    - über SelectionStrategy werden daraus folgende Sets gefüllt: informationObjectsForFlow, informationObjectsForTeaser
- Instanzvariable: flowItemsGroup - hier werden FlowVisualItems eingefügt

In FlowView.onInit() wird eine neue VisualGroup instantiiert und der Instanzvariable flowItemsGroup zugewiesen.

Weiterhin werden in onInit() über updateDataList() die darzustellenden Informationspartikel aus dem Mashup geladen. Siehe hierzu auch Informationsobjekte - Auswahl

onUpdate() wird regelmäßig aufgerufen (siehe folgenden Abschnitt zu Animation).

In onUpdate() passiert folgendes:

- Falls noch nicht maximal viele VisualItems unterwegs sind, füge ein Neues hinzu

```
if (this.flowItemsGroup.getChildren().size() < this.maxNumberFlowingItems) {
    FlowVisualItem<?, ?> item = this.addFlowVisualItem();
```

Zu beachten bei der Einbindung externer Ressourcen: Die Klasse FXView hält auf Instanz-Ebene einen org.sociotech.cmf3.configuration.ThemeResourceLoader. Diese Hilfsklasse ermöglicht das Laden von Ressourcen aus Themes, d.h. aus Unterordnern von resources/themes. Eine Referenz auf diesen ThemeResourceLoader muss an alle Unterobjekte, welche Theme-spezifische Ressourcen laden (z.B. FXML- oder CSS-Dateien), weitergereicht werden. Dafür hat VisualComponent eine Instanzvariable themeResources, welche standardmäßig null ist. Sie sollte auf den ThemeResourceLoader zeigen, falls die Komponente von externen, Theme-spezifischen Ressourcen Gebrauch macht. Für weitere Details siehe auch weiter unten "Layout im Kleinen - render()".

Wichtige Klassen (von oben nach unten jeweils Ableitungsbeziehung):

Klasse	Beschreibung
javafx.scene.Node	
javafx.scene.Group	
org.sociotech.cmf3.components.VisualComponent	Die wichtigste ...
org.sociotech.cmf3.components.VisualItem	
org.sociotech.cmf3.components.FXMLVisualItem	Unterstützung der Generierung von view und controller aus FXML-Dateien
org.sociotech.cmf3.view.flow.components.FlowVisualItem	aktuell nichts Neues hier ...

Animation

In der Basisklasse **CommunityMirror** wird Mittels eines JavaFX TimelineBuilders ein JavaFX Timeline-Objekte erzeugt und dann .play() darauf aufgerufen. Die handle()-Funktion ruft dann auf allen Controller-Objekten und View-Objekten die Funktion update() auf, welche wiederum die Funktion onUpdate() aufruft.

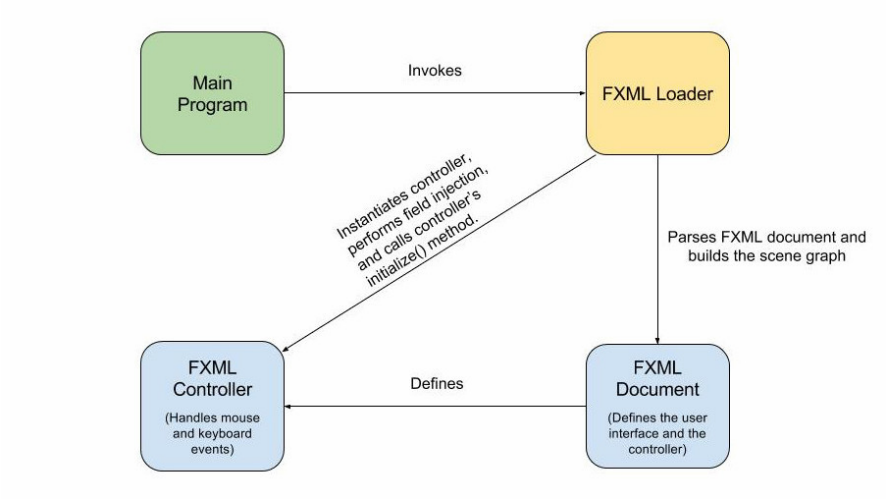
Animationen von VisualComponent-Objektion sind aktuell folgendermaßen realisiert:

- In **VisualComponent** können physicsBehaviors registriert werden (z.B. die DriftBehavior zum Weiterbewegen nach einem Drag)
- in onUpdate wird dann bei jedem registrierten Behavior-Objekt die Funktion applyTo() aufgerufen um das Verhalten auf das Objekt anzuwenden

Layout im Kleinen (einzelne Informationspartikel -> als VisualItem)

Die **Model-View-Controller-Idee** wird folgendermaßen umgesetzt:

- Das Model sind die Informationspartikel (also die Item-Objekte von CommunityMashup)
- Die View sind die Instanzen der FlowVisualItem-Objekte, deren konkretes Layout in FXML-Dateien definiert ist (die Auswahl der FXML-Datei erfolgt in FXMLVisualItem).
- Der Controller ist das FXMLController-Objekt, das in der init()-Methode des FlowVisualItem erzeugt wird (bzw. genauer in der von der init()-Methoden aufgerufenen Methode loadViewAndController()) - dieses Objekt erhält eine Referenz auf das Model und versorgt die View mit Daten - und wird aufgerufen, wenn die View Interaktionen handhabt



Hinweis: Die MVC-Umsetzung im CMF3 ist nicht "ganz sauber", denn die Klasse FlowVisualItem definiert Callbacks für Touch und MouseEvents etc - übernimmt also einige Aufgaben eines Controllers.

Informationspartikel (also Objekte aus dem Mashup) werden, wie oben ausgeführt, hauptsächlich in der Funktion **FlowView.onUpdate()** in den Flow eingefügt. Dazu wird die Funktion **FlowView.addFlowVisualItem()** verwendet.

Informationspartikel können dabei verschiedene Detailgrade haben. Aktuell werden im FlowView die folgenden zwei Detailgrade genutzt:

- **inPreview:** "normale" Darstellung eines IPs im Flow, hierbei v.a. Schwerpunkt auf Bildmaterial für "User Attraction" und das Wecken von ausreichendem Interesse (kein "grau in grau")
- **inDetail:** Detailanzeige, einzige Darstellungsform bei der wirklich alle Inhalte angezeigt werden, ggf. auch Anhänge (z.B. PDFs von Papern = Content) zum "Durchblättern".

In der Funktion **FlowView.addFlowVisualItem()** wird hauptsächlich die Funktion **FlowView.createFlowVisualItem**() aufgerufen. Diese Funktion erzeugt eine Instanz eines FlowVisualItem und ruft dann die init()-Methode dieser Klasse auf.

In der init()-Methode von FlowVisualItem passiert hauptsächlich der Aufruf der Funktion **FXMLVisualItem.loadViewAndController()**.

In loadViewAndController() wird folgendes gemacht

- Bestimmung des Namens der FXML-Datei aus VisualState (preview, detail), Typ des Content Items sowie MetaTags des Content Items: Konkret wird über die Funktion getTemplateFileName() über die entsprechende Funktion in FlowView geprüft, zu welchen Meta-Tags es spezielle Template-Dateien gibt (Format "itemtype-metatag.XXX") und diese genutzt wenn das MetaTag gesetzt ist. Template-Dateien zu den Item-Type (content, person, organisation) sind immer vorhanden.
- Aufruf der Funktion load() im FXMLLoader mit dem Namen der FXML-Datei (aus den ThemeRessources)
- Das Hauptobjekt in der FXML-Datei ist normalerweise eine AnchorPane - z.B.: <AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="200" prefWidth="200" fx: controller="org.sociotech.cmf3.view.flow.components.ContentItemController" xmlns:fx="http://javafx.com/fxml">
- Hier wird also auch die Klasse des ItemControllers definiert
- Ergebnis des Ladens ist dann ein AnchorPane-Objekt, welches über .this.getChildren().add() dem FlowVisualItem hinzugefügt wird, sowie ein ItemController-Objekt (über loader.getController() gelesen und dann in einer Instanzvariable des FXMLVisualItem gespeichert)
- Auf dem Controller-Objekt wird dann die Methode bindData() aufgerufen und dabei eine Referenz auf das Modell (d.h. das CommunityMashup Item-Objekt) übergeben - in der Methode werden die Daten des Modells an die Objekte im durch das FXML erzeugten Scene-Graph gebunden.
- In bindData() wird für die WebView, welche für die Darstellung des Detail-Views verwendet wird, noch ein Template-File geladen und interpretiert (Funktion generateDetailDescription()). Hierzu wird das Toolkit FreeMarker zum Template-Parsing genutzt - aktuell wird FreeMarker eine Referenz auf das passende Item-Objekt (also Content, Person oder Organisation) mitgegeben.

VisualItems, welche Theme-spezifische externe Ressourcen einlesen, sollten dafür wenn möglich den ThemeResourceLoader verwenden.
In den implementierten FlowVisualItems ist der Ablauf derzeit wie folgt:

1. Es wird überprüft, ob das FlowVisualItem eine gültige Referenz auf einen ThemeResourceLoader enthält (da das Setzen dieser Variable durch das View optional ist - siehe oben).
2. Existiert ein ThemeResourceLoader, wird die Ressource durch dessen Schnittstelle geladen, d.h. aus dem jeweiligen Unterverzeichnis unterhalb von resources/themes. Das aktuelle Theme kann in der Konfiguration geändert werden.
3. Existiert kein ThemeResourceLoader, wird die externe Ressource aus einem fest im Code angegebenen Verzeichnis geladen, z.B: "resources /flow", und es wird eine entsprechende Warnung ins Log gegeben. Langfristig, wenn alle VisualItems erfolgreich auf die Nutzung des ThemeResourceLoaders umgesattelt sind, kann dieser Code-Pfad und die dazugehörigen Ressourcendateien entfernt werden.

CSS: Bei der Definition von JavaFX-Komponenten in den FXML-Dateien kann CSS verwendet werden. Dafür wird ein in FlowView.onInit() geladenes "theme.css" genutzt. Weitere CSS-Dateien können bei Bedarf direkt in FXML-Dateien nachgeladen werden.