

Praxisbericht Nr. 3

Prüfling (Name) Niels Gundermann

Matrikel-Nr. 5023

Studiengang, Zenturie Wirtschaftsinformatik, I11b

Thema Prototypische Implementierung eines JavaFX-Channels zur Integration ins Multichannel-Framework der deg

Ausbildungsbetrieb data experts gmbH

Prüfer/in Dr. Stephan Neuthe

E-Mail der/ des Prüfers/in stephan.neuthe@data-experts.de

Datum der Themenausgabe 22.03.2013

Datum der Abgabe 18.4.2013

fristgerechte Abgabe¹

Ja ☒

Nein ☐

Sperrvermerk²

Ja ☐


Nein ☒

bestanden³

Ja ☒

Nein ☐

Datum 4.12.2014

Unterschrift Prüfer/in 

¹ Die Bearbeitungsdauer für Praxisberichte soll vier Wochen nicht überschreiten (vgl. Merkblatt Praxisberichte). Wenn der Praxisbericht nicht innerhalb der vereinbarten Frist abgegeben wurde, dann gilt die Studienleistung als nicht bestanden.

² Der Praxisbericht enthält interne, vertrauliche Daten, die nicht zur Weitergabe an Dritte bestimmt sind.

³ Als Bewertung kommen nur „bestanden“ oder „nicht bestanden“ in Betracht. Prüfer nutzen hierfür das beigefügte Bewertungsschema.

Eidesstattliche Erklärung des Studenten / der Studentin

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder von mir noch von jemand anderem als Prüfungsleistung vorgelegt. Der von mir angegebene Prüfer hat die Arbeit zur Kenntnis genommen und als „bestanden“ bewertet.

Datum **04.12.2014**
.....

Unterschrift Student/in


.....

Praxisbericht: Prototypische Implementierung eines JavaFX-Channels zur Integration ins Multichannel-Framework der deg

Niels Gundermann

18. April 2013

Inhaltsverzeichnis

1	Einleitung	4
2	Warum sollte Swing durch JavaFX abgelöst werden?	4
2.1	Vergleich: Swing - JavaFX	5
3	Beispielimplementation	7
3.1	Das Menü - MenuBar und ToolBar	8
3.2	SplitPanels	9
3.3	TreeViews	10
3.4	Tabellen	10
3.5	Benötigte Events	11
3.6	Layout-Variationen	12
3.7	Probleme	13
4	Fazit	14

Abbildungsverzeichnis

1	Vergleich: Swing - JavaFX	16
2	Überblick der Beispielimplementation	17
3	Scene - Graph	17

1 Einleitung

Im dritten Quartal 2011 erschien das von Oracle angekündigte JavaFX 2.0. Für viele Softwareentwickler stellen sich die Frage, ob es praktisch ist, ihre Programme auf JavaFX umzustellen, und neue Programme gleich mit Hilfe von JavaFX zu implementieren. Das gilt auch nicht nur für Webentwickler. JavaFX ist zwar ein Framework für Plattformübergreifende Rich Internet Applications, aber dennoch in Offline-Anwendungen durch das JFX-Pane verfügbar.

Also, with the new JFXPane component that allows embedding of JavaFX applications in Swing, it is easy to begin incorporating JavaFX elements in existing web applications. (Stephen Chin¹)

Somit können die Entwickler sogar überlegen, ob sie Swing ganz vernachlässigen wollen. Aber warum sollte die Entwickler überhaupt Swing durch JavaFX ersetzen wollen? Was hält JavaFX überhaupt bereit?

Solche Fragen sollen in dieser Arbeit mit Bezug auf die deg beantwortet werden. Ferner wird an einem Profil C/S-ähnlichen Beispiel gezeigt, was mit JavaFX möglich ist. Dabei ist vorab zu erwähnen, dass sich die Ähnlichkeit zu Profil C/S nur auf die GUI beschränkt und die elementarsten Eigenschaften beinhaltet, die für Profil C/S benötigt wird.

Ein weiterer Schwerpunkt wird auf den Unterschieden zwischen Swing und JavaFX, sowie den Vor- und auch Nachteilen beider Frameworks liegen. Vor dem Fazit wird auch auf Probleme eingegangen, die während des Erstellens der angefügten Beispielimplementation zutage kamen.

Das Fazit soll zeigen, welche Vorteile die deg aus JavaFX ziehen könnte und ob es sich für sie lohnt, sich mit diesem Thema weiter zu beschäftigen

2 Warum sollte Swing durch JavaFX abgelöst werden?

Ende Dezember 2008 wurde das erste Entwicklerkit für das damalige JavaFX 1.0 veröffentlicht. In diesen Versionen wurden die Anwendungen noch mithilfe von JavaFX Script programmiert. Das Ziel war damals GUIs schnell und allgegenwärtig bereitzustellen. Das Resultat war jedoch, dass es viele Java-Entwickler als umständlich empfanden, mit zwei unterschiedlichen Sprachen zu arbeiten, wo eine einzige genügen würde. Desweiteren gab es vor allem in dieser Version große Probleme mit der Performance, was letztendlich dazu

¹Stephen Chin: Java Technology Evangelist at Oracle

geführt hat, dass JavaFX von vielen nicht so angenommen wurde, wie man eigentlich gehofft hatte.

Im dritten Quartal 2011 veröffentlichte Oracle die Version 2.0. Damit waren die größten Probleme behoben. Zum einen war JavaFX-Script nicht mehr notwendig, da sich der gesamte Code auch in Java schreiben lies. Und zum Anderen wurde das Rendering so optimiert, dass es nunmehr mit Hardware-Unterstützung durchgeführt wird, was die Performance stark verbesserte.

Bei Swing wurden Die letzten signifikanten Verbesserungen 2002 veröffentlicht, was alleine schon ein Indiz dafür ist, dass diese API nicht weiter gepflegt, oder optimiert wird. Bei diesen Änderungen ging es damals ebenfalls um die Verbesserung der Hardware-Unterstützung. Erweiternde Frameworks wie Spring Richclient, welches die Integration in das Spring Framework bietet, wird ebenfalls seit 2009 nicht mehr weiterentwickelt.

Seit JavaFX 2.2 - erschienen im August 2012 - ist das Framework in Java SE7 integriert und kann somit zunehmend vorausgesetzt werden. Noch dazu kommt, dass diejenigen, die bei Oracle für die Verbreitung von JavaFX verantwortlich sind, in ihren Aussagen immer wieder ankündigen, dass JavaFX der Nachfolger, bzw. ein guter Nachfolger für Swing ist.

With the 2.0 release, JavaFX finally has the performance, Java language support, and business focused controls to be a worthy successor to Swing. This is good news to those companies who have a heavy investment in Swing, because they have a path forward to continue developing mission critical business applications on the highly robust JVM platform. (Stephen Chin)

Demnach ist davon auszugehen, dass Oracle in Zukunft die Ablösung von Swing durch JavaFX anstrebt. Sicherlich ist an eine zeitnahe Ablösung nicht zu denken, da JavaFX noch zu unbekannt ist. Trotzdem ist es, in Betrachtung dessen, dass bei Swing keine großartigen Neuerungen mehr zu erwarten sind und zunehmend daran gearbeitet wird JavaFX in Swing zu integrieren, für die eigene Weiterentwicklung auf GUI-Ebene notwendig, sich mit JavaFX auseinanderzusetzen. Ein direkter Vergleich zwischen JavaFX und Swing wird im nächsten Kapitel vorgestellt.

2.1 Vergleich: Swing - JavaFX

Da Swing allgemein bekannt ist, wird dieser Vergleich ein wenig verdeutlichen, was von JavaFX zu erwarten ist. Dabei sollte bedacht werden, dass Oracle ambitioniert ist, JavaFX weiterzuentwickeln. Aufgrund dessen ist dieser Vergleich auf Seiten von JavaFX

noch nicht endgültig.

Grobe Gemeinsamkeiten lassen sich vor allem in der Implementierung nachvollziehen. Swing- und JavaFX-Programmen können komplett in Java geschrieben werden. Somit können Java-Entwickler weiterhin ihre favorisierten Entwicklungsumgebungen und Programmiersprachen beibehalten.

Weiterhin bieten Swing und JavaFX eine große Auswahl von Layouts und anderen GUI-Elementen, die es ermöglichen, das Programm ansprechend zu gestalten. Auch selbstgezeichnete GUI-Komponenten können sowohl in Swing, als auch in JavaFX erstellt werden. JavaFX bietet darüber hinaus das FXML. Diese XML-basierte Dateien bieten eine einfache Möglichkeit komplexe GUIs zu entwickeln. Dabei werden alle GUI-Elemente in diese Datei geschrieben, welche dann innerhalb von Java nur in das JavaFX-Objekt eingebunden werden muss. Um das noch weiter zu vereinfachen, wurde der SceneBuilder entwickelt. Mit diesem Tool lassen sich die FXML-Dateien automatisch zusammenbauen.

Beide Frameworks sprechen ihre Komponenten über Events an, die ähnlich implementiert sind.

Eine weitere Gemeinsamkeit ist das Plattform-unabhängige Entwickeln bzw. die Portabilität durch das Java Runtime Environment. Damit ist gewährleistet, dass die Programme auf 97% der Betriebssystemrechner auf der Welt lauffähig wären.

Was die Implementation angeht ist noch zu erwähnen, dass JavaFX bezogen auf den Top-Level-Container einfachen zu händeln ist, da man es nur mit einem Container zu tun hat. Swing stellt dahingegen 3 verschiedene Top-Level-Container zur Verfügung.

Swing hat einen großen Vorteil gegenüber JavaFX und das ist der Bekanntheitsgrad. Viele Entwickler sind mit der Struktur vertraut und wissen, wie sie mit Swing umgehen müssen. Wohingegen JavaFX noch nicht ausgereift ist und deshalb in größeren Projekten noch nicht verwendet wird.

In JavaFX ist weiterhin eine Web rendering engine integriert. Damit ist es möglich Anwendungen zu entwickeln, in denen bestehende Webanwendungen integriert werden können. Das ermöglicht einen nahtlosen Übergang zwischen den Java-Funktionen und Funktionen, welche die Dynamik der Web-Technologien widerspiegeln.

Ein sicherlich aus eben diesen Web-Technologien stammendes Feature, ist die Media engine. Die eine Integration von Video- und Audio-Inhalte in das Programm ermöglicht.

In diesem Bereich ist jedoch nicht nur die eine Richtung zu betrachten. JavaFX unterstützt genauso wie Swing Webanwendungen (Applets). Zusätzlich ist es durch JavaFX aber auch

möglich, sich mit einem Touchscreen ohne Cursor zu navigieren.

Weiterhin ist JavaFX in der Lage die Hardwarebeschleunigung des Rechners anzusprechen. Dadurch erreicht man auf Grafik-Ebene eine weitaus höhere Performance. Mit Swing machen sich oft Performanceprobleme bemerkbar.

Einen weiterer Nachteil, der bei dem Einsatz von Swing auftritt, ist das starre Design. In der Regel ist immer zu erkennen, dass eine Application mit Swing erstellt wurde, da die Komponenten nicht groß verändert werden können. JavaFX ist da variabler.

Wie zu Beginn schon erwähnt, bietet JavaFX genauso wie Swing viele Möglichkeiten die GUIs entsprechend zu gestalten. JavaFX ist jedoch in der Lage, diese Elemente über Cascading Style Sheets zu beeinflussen. Das erleichtert ein durchgehendes Design.

Abbildung 1 zeigt eine übersichtliche Darstellung dieses Vergleichs.

3 Beispielimplementation

Die Beispielimplementation soll aufzeigen inwiefern JavaFX die Funktionen bereitstellt, die von der deg benötigt werden. Weiterhin werden die wichtigsten Komponenten und ihre Implementation kurz erklärt. Zum Nachvollziehen des visuellen Ergebnisses befindet sich das Programm im Anhang. Da dieses Beispiel nur der Vorstellung dienen soll, ist die GUI minimal gehalten.

Abbildung 2 zeigt die Startansicht und die Unterteilung in bestimmte Bereiche, auf die im weiteren Verlauf Bezug genommen wird.

Jede JavaFX-Anwendung muss von *javafx.application.Application* erben. Dadurch wird eine Methode *start(Stage primaryStage)* implementiert. Diese Methode ist der Ausgangspunkt einer jeden GUI, die mit JavaFX entwickelt wird. Die *Stage* ist der Top-Level-Container in JavaFX. Dieser muss vorm Ausführen der *launch*-Methode initialisiert werden.

Der Top-Level-Container in JavaFX kann alleine keine weitere GUI aufbauen. Er benötigt Elemente, die sich anders strukturieren lassen. Hierzu wird eine *Scene* initialisiert, die sich, wie in Abbildung 3 ersichtlich, als Baum betrachten lässt. Als Wurzelement wird die *Group* vorgeschlagen. Der Vorteil ist, dass alle direkten Kinder dieses Elementes neu ausgerichtet werden, falls es zu einer Transformation der Komponente kommt (*Scene*). Es ist allerdings auch möglich anstelle der *Group* ein konkretes Layout zu setzen.

Im Anschluss daran wird das Layout für die GUI über ein *Pane* definiert. Der Unterschied

zu Swing liegt darin, dass man bei JavaFX eine komplette GUI-Komponente initialisiert und das Layout nicht explizit setzen muss. Da die *Scene* als Baum organisiert ist, sind die nachfolgenden Methodennamen gut nachvollziehbar.

```
1 public class VertragsMappeFXController extends VertragsMappeFXGui{
2
3     public VertragsMappeFXController() {
4         try {
5             super.start(new Stage());
6         } catch (Exception e) {
7             e.printStackTrace();
8         }
9
10        ...
11 }
12
13 public class VertragsMappeFXGui extends Application {
14
15     Group root;
16     Scene scene;
17
18     @Override
19     public void start(Stage primaryStage) throws Exception {
20         root = new Group();
21         scene = new Scene(root, 800, 500);
22
23         BorderPane borderPane = new BorderPane();
24         root.getChildren().add(borderPane);
25
26         primaryStage.setScene(scene);
27         primaryStage.show();
28     }
29     ...
30 }
```

3.1 Das Menü - MenuBar und ToolBar

Die Menüzeile ist wie in Swing eine eigene Komponente in JavaFX, die wie alle anderen Komponenten aus bestimmte Bereiche eines Panes gelegt werden kann. Innerhalb der MenuBar sind nur die einzelnen Menüs und die darin enthaltenden Felder zu konfigurieren. Folgendes Beispiel verdeutlicht, wie einfach diese Problematik für einen Menüpunkt gelöst wird:

```
1 private MenuBar initMenu() {
2     MenuBar menu = new MenuBar();
3
4     Menu bearbeitung = new Menu("Bearbeiten");
5     menu.getMenus().add(bearbeitung);
6
7     MenuItem drucken = new MenuItem("Drucken");
8     MenuItem seitenansicht = new MenuItem("Seitenansicht");
9
10    bearbeitung.getItems().addAll(drucken, seitenansicht, schliessen);
11    ...
12
13    return menu;
14 }
```

Dasselbe gilt für die Toolbar, welcher man jedoch nur Schaltflächen und sogenannte Separatoren, die als Trennlinien zwischen den Schaltflächen visualisiert werden, hinzufügen kann.

```

1 private Toolbar initToolbar() {
2     Toolbar toolbar = new Toolbar();
3
4     Image imageDrop = new Image("/Icons/TbCopy.gif");
5     Image imagePrint = new Image("/Icons/TbPrint.gif");
6     Image imageOrg = new Image("/Icons/TbOriginal.gif");
7     Image imageGetOrg = new Image("/Icons/TbKopie.gif");
8     Image imageHelp = new Image("/Icons/TbHelp.gif");
9     Button btDrop = new Button(null, new ImageView(imageDrop));
10    btDrop.getStyleClass().add("barbutton");
11    Button btPrint = new Button(null, new ImageView(imagePrint));
12    btPrint.getStyleClass().add("barbutton");
13    btPrint.setDisable(true);
14    Button btLossOrg = new Button(null, new ImageView(imageOrg));
15    btLossOrg.getStyleClass().add("barbutton");
16    Button btGetOrg = new Button(null, new ImageView(imageGetOrg));
17    btGetOrg.getStyleClass().add("barbutton");
18    btGetOrg.setDisable(true);
19    Button btHelp = new Button(null, new ImageView(imageHelp));
20    btHelp.getStyleClass().add("barbutton");
21
22    toolbar.getItems().addAll(btDrop, new Separator(), btPrint,
23        new Separator(), btLossOrg, btGetOrg, new Separator(), btHelp);
24
25    return toolbar;
26 }

```

3.2 SplitPanes

Auch in Bezug auf das Teilen eines bestimmten Bereichs des Fensters, ist JavaFX sehr an Swing angelehnt. Horizontale und vertikale Unterteilungen sind möglichen, genauso wie bestimmte Voreinstellungen, was die Größe der unterteilten Bereiche angeht. Das Beispiel zeigt zwei *Splitpanes*, die unterschiedlich orientiert sind und spezielle Größeneinstellungen haben:

```

1 private HBox initSplitPane(Group root) {
2     HBox splitPaneBereich = new HBox();
3
4     SplitPane horizontalerSplit = new SplitPane();
5     horizontalerSplit.setDividerPositions(0.3);
6
7     SplitPane vertikalerSplit = new SplitPane();
8     vertikalerSplit.setOrientation(Orientation.VERTICAL);
9
10    VBox obereBereich = initDocumentBaum();
11    VBox untererBereich = initVerweiseBaum();
12
13    vertikalerSplit.getItems().add(obereBereich);
14    vertikalerSplit.getItems().add(untererBereich);
15
16    BorderPane right = new BorderPane();
17    HBox rightArea = initBearbeitungInhalt();
18    right.setCenter(rightArea);
19    VBox leftArea = new VBox();
20    leftArea.getChildren().add(vertikalerSplit);

```

```

21
22         horizontalerSplit.getItems().addAll(leftArea, right);
23         horizontalerSplit.prefWidthProperty().bind(scene.widthProperty());
24
25         splitPaneBereich.getChildren().add(horizontalerSplit);
26         return splitPaneBereich;
27     }

```

Über *setDividerPositions()* wird die Standardgrößeneinstellung des *Splitpanes* prozentual festgelegt. Die Trennlinie teilt den Bereich im Beispiel in einen Bereich, dessen Größe 30% des Ganzen ausmachen und einen weiteren Bereich, der die anderen 70% einnimmt. Die GUI-Elemente können nicht direkt in bestimmte Bereiche des *Splitpanes* eingefügt werden. Das erste Element, welches hinzugefügt wird, findet sich im ersten Bereich des *Splitpanes* wieder. Bei einer horizontalen Orientierung ist dieser der linke Bereich und bei einer vertikalen Orientierung der obere Bereich.

3.3 TreeViews

Die Baumstrukturen sind weiterhin wie in Swing sehr einfach gehalten. Sie werden als *TreeView* initialisiert. Dieser *TreeView* werden dann *TreeItems* hinzugefügt, denen man dann wieder *TreeItems* hinzufügen wird. So ist es möglich Eltern- und Kinderelemente zu erstellen. Im Beispiel werden alle registrierten Dokumente, als Kinder an ein Wurzelement des Baumes gehängt:

```

1 ...
2         TreeView<String> documentTree = new TreeView<String>();
3         TreeItem documentRoot = new TreeItem();
4         documentRoot.setValue(currentMappe.getFp());
5         documentTree.setRoot(documentRoot);
6
7         appendDocuments(currentMappe, documentRoot);
8
9         documentBox.getChildren().add(documentTree);
10        ...
11
12 private void appendDocuments(Document doc, TreeItem treeItem) {
13     for (Document children : doc.getChildren()) {
14         TreeItem documentItem = new TreeItem();
15         documentItem.setValue(children.getTitel());
16         treeItem.getChildren().add(documentItem);
17         appendDocuments(children, documentItem);
18     }
19
20 }

```

3.4 Tabellen

Die Tabellen in JavaFX werden als *TableView* initialisiert. An dieser Komponente kann man die Spalten als Kinder anhängen. Diese Kinder werden wiederum als *TableColumn*

initialisiert. Dabei wird ihnen die Bezeichnung als Parameter übergeben. Im Anschluss daran muss an den Spalten eine *CellValueFactory* gesetzt werden. Darüber werden die Werte, die in der Tabelle stehen sollen, den Spalten zugewiesen. Das funktioniert wie ein Dictionary wobei der String, den man der *PropertyValueFactory* übergibt, das Schlüsselwort ist.

```

1      TableView<TeilvorgaeneTableData> table = new TableView<TeilvorgaeneTableData>();
2      TableColumn vorgangCol = new TableColumn("Vorgang");
3      vorgangCol.setCellValueFactory(
4      new PropertyValueFactory<TeilvorgaeneTableData, String>("vorgang"));
5      table.getColumns().addAll(vorgangCol);

```

Die Werte in den Zellen werden an der *TableView* gesetzt. Dabei handelt es sich um eine *ObservableList*, die bestimmte Objekte enthalten kann. Sind an diesen Objekten Variablen des Typs *SimpleStringProperty* gesetzt, deren Bezeichner mit den Schlüsseln der *PropertyValueFactory* übereinstimmen, werden die Werte der Variablen gemäß der Definition in den Spalten, in die Tabelle eingefügt. Dabei ist darauf zu achten, dass der Getter der Variable die *SimpleStringProperty* als *String* ausgeben sollte und eine mögliche *NullPointerException* abfangen muss.

```

1 ...
2 private void initLines(TableView table) {
3     ObservableList<TeilvorgaeneTableData> vorgaenge = FXCollections.observableArrayList();
4     for(Document teilvorgang : teilvorgaenge){
5         vorgaenge.add(new TeilvorgaeneTableData(teilvorgang));
6     }
7     table.setItems(vorgaenge);
8 }
9 ...
10 ...
11
12 public class TeilvorgaeneTableData {
13
14     private SimpleStringProperty vorgang;
15
16     public TeilvorgaeneTableData(Document teilvorgang) {
17         this.TeilvorgaeneTableData((Auszahlung) teilvorgang);
18     }
19
20     private void TeilvorgaeneTableData(Auszahlung teilvorgang) {
21         this.vorgang = new SimpleStringProperty(teilvorgang.getTitel());
22     }
23
24     public String getVorgang() {
25         if(vorgang == null)
26             return "";
27         return vorgang.get();
28     }
29 }

```

3.5 Benötigte Events

Als »benötigt« werden die Events betrachtet, welche die Steuerung des Beispiels ermöglichen. Das beschränkt sich auf Reaktionen auf das Betätigen von Schaltflächen und auf das Se-

lektieren von Tabelleneinträgen.

Solange es sich bei JavaFX um einfache Schaltflächen handelt, werden dafür `EventHandler` verwendet. Ähnlich wie bei Swing werden diese Handler einfach an den entsprechenden GUI-Elementen gesetzt:

```
1 private void initToolBarController() {
2
3     final Button btGetOrg = ((Button) super.getToolBar().getItems().get(5));
4     final Button btLossOrg = ((Button) super.getToolBar().getItems().get(4));
5     final Label lbOriginal = ((Label) super.getStatusbarOriginalLabel());
6
7     btGetOrg.setOnAction(new EventHandler<ActionEvent>() {
8
9         @Override
10        public void handle(ActionEvent arg0) {
11            btLossOrg.setDisable(false);
12            btGetOrg.setDisable(true);
13            lbOriginal.setText("");
14        }
15    });
16
17    btLossOrg.setOnAction(new EventHandler<ActionEvent>() {
18
19        @Override
20        public void handle(ActionEvent arg0) {
21            btLossOrg.setDisable(true);
22            btGetOrg.setDisable(false);
23            lbOriginal.setText("Original");
24        }
25    });
26
27 }
```

Bei Tabellen ist es ebenfalls sehr einfach. Nur das hierbei nicht mit der Methode `setOnAction(EventHandler)` gearbeitet wird, sondern mit `setOnMouseClicked(EventHandler)`.

Die jeweils selektierte Zeile wird über das *SelectionModel* der *TableView* ausgelesen:

```
1 table.setOnMouseClicked(new EventHandler<Event>() {
2
3     @Override
4     public void handle(Event arg0) {
5         details.updateLines(vertragsblatt.getZuwendungen().
6             get(table.getSelectionModel().getSelectedIndex()));
7     }
8 });
```

3.6 Layout-Variationen

Das Layout erinnert auch sehr an Swing. Es ist auch hier möglich mit *BorderLayout*/*BorderPane* und co. zu arbeiten. Interessant ist, dass JavaFX das *GridLayout* und das *GridBagLayout* aus Swing zusammenfasst. Das *GridPane* bietet die Funktionen von beiden an².

JavaFX biete darüber hinaus die Möglichkeit die Elemente mittels Cascading-Style-Sheets

²JavaFX 2 Dokumentation: [<http://docs.oracle.com/javafx/2/api/javafx/scene/layout/GridPane.html>]

zu gestalten. Neben den Standardfunktionen von CSS 2.1 werden von JavaFX weitere CSS-Funktionen bereitgestellt. Um diese Funktion nutzen zu können, muss die CSS-Datei eingebunden werden:

```
1 ...
2         String css = "layout.css";
3         ObservableList<String> cssStyle = loadSkin(css);
4         scene.getStylesheets().clear();
5         scene.getStylesheets().addAll(cssStyle);
6 ...
7 protected final ObservableList<String> loadSkin(String cssFileName) {
8         ObservableList<String> cssStyle = FXCollections.observableArrayList();
9         cssStyle.addAll(getClass().getResource(cssFileName).toExternalForm());
10        return cssStyle;
11    }
```

Den GUI-Elementen weißt man dazu eine CSS-Klasse zu und kann sie dementsprechend in der CSS-Datei modifizieren:

```
1 ...
2         Button btHelp = new Button(null, new ImageView(imageHelp));
3         btHelp.getStyleClass().add("barbutton");
4 ...
5 //CSS-File:
6 ...
7 .barbutton{
8 -fx-border-color: rgba(225, 255, 255, .80);
9 -fx-border-radius: 8;
10 -fx-padding: 6 6 6 6;
11 }
12 ...
```

3.7 Probleme

Bei der Arbeit mit JavaFX kam es immer wieder zu Problemen mit der Größe von Kinderelementen. Besonders bei der Verwendung von SplitPanes wurden die Größeneinstellungen der Elternelemente nicht übernommen, was dazu führte, dass die Kinderelemente das Elternelement meist nicht ausfüllten. Die Größeneinstellungen der einzelnen Elemente sind als *prefWidthProperty* und *prefHeightProperty* festgesetzt. Es ist in solchen Fällen, wenn die gewünschte Größe nicht als Standard gesetzt ist, weil sie zum Beispiel von den Elementen im Innern des betrachteten Elementes abhängt, bietet es sich an, die Größeneinstellung des Elternelementes zu übernehmen:

```
1         ...
2         HBox splitPane = initSplitPane(root);
3         ...
4         borderPane.setCenter(splitPane);
5         borderPane.prefHeightProperty().bind(scene.heightProperty());
6         borderPane.prefWidthProperty().bind(scene.widthProperty());
7         ...
8
9 private HBox initSplitPane(Group root) {
10        HBox splitPaneBereich = new HBox();
11
12        SplitPane horizontalSplit = new SplitPane();
```

```

13         horizontalerSplit.setDividerPositions(0.3, 0.7);
14         ...
15         horizontalerSplit.getItems().addAll(leftArea, right);
16         horizontalerSplit.prefWidthProperty().bind(scene.widthProperty());
17
18         splitPaneBereich.getChildren().add(horizontalerSplit);
19         return splitPaneBereich;
20     }

```

4 Fazit

Wer Erfahrung mit Swing hat, dem fällt auf, dass es bezogen auf die Syntax keine großen Unterschiede zwischen beiden Frameworks gibt. Die Umstellung auf die sich die Entwickler einlassen müssen, fällt demnach nicht sehr groß aus, womit dieses Gegenargument für die Verwendung von JavaFX an Gewichtung verliert. Weiterhin wurde Swing von seinen ursprünglichen Entwicklern abgeschrieben und wird nicht weiter gepflegt. Daraus folgt, dass man mit einem Framework arbeitet, für das es keinerlei Support gibt. Um das Know-How in Bezug auf Swing zu erweitern, gibt es somit keinerlei oder nur veraltete Quellen, was die Effizienz deutlich beeinträchtigt. Von daher ist es von Vorteil Know-How im Bereich JavaFX aufzubauen.

Es steht außer Frage, dass eine Umstellung ein hoher Aufwand ist. Dieser Aufwand wird auch noch zunehmen, falls man sich dazu entscheidet bisherige in Swing geschriebene GUI auf JavaFX umzustellen. Noch dazu muss die das neue GUI-Framework in ihr Multichannel-Framework integrieren. Bezogen auf eine reine zukunftsorientierte Umstellung, sprich zukünftig neue Projekte mit anderen Frameworks zu entwickeln, ist meiner Meinung nach kein großes Problem. Auch wenn Schulungen für die Entwickler nötig werden, fördert das nur den Erfolg des Projektes. Problematisch ist die zweite Stufe: Swing zu JavaFX umwandeln. Hierzu wurde schon erwähnt, dass sich JavaFX in Swing integrieren lässt. Daher ist es relativ einfach, die bestehenden Komponenten in den GUIs zu verbessern, oder zu aktualisieren. Auch gibt es Hinblick auf Umstellungen von Swing auf JavaFX bereits Firmen³, welche an Tools arbeiten, die es ermöglichen sollen, automatisch die Swing-Komponenten durch JavaFX-Komponenten zu ersetzen. Solche Tools werden aber spätestens an der Integration in den Multichannel scheitern.

Somit wäre der nächste Schritt nach Möglichkeiten zu suchen JavaFX in das Multichannel-Framework einzubinden. Nicht um voranging alle Swing-Komponenten umzustellen, sondern um zukünftig mit JavaFX entwickeln zu können.

³Open Knowledge

Quellen

Carl Dea: JavaFX 2.0 Introduction by Example, 2012, Apress

Oracle: JavaFX - The Rich Client Platform

URL: <http://www.oracle.com/technetwork/java/javafx/overview/index.html> [eingesehen am 29.01.2013]

Oracle-Blog: JavaFX URL: <https://blogs.oracle.com/javafx/> [eingesehen am 29.02.2013]

Netzwelt: JavaFX 2.0 URL: <http://www.netzwelt.de/news/88783-javafx-2-0-flash-konkurrent-open-source.html> [eingesehen am 30.01.2013]

Oracle-Dokumentation: Swing Enhancements

URL: <http://docs.oracle.com/javase/6/docs/technotes/guides/swing/6.0/index.html#Parentless> [eingesehen am 28.01.2013]

IT-Republic: Swing Ade: Die Zukunft heißt JavaFX

URL: <http://it-republik.de/jaxenter/news/Swing-Ade-Die-Zukunft-heisst-JavaFX-062432.html> [eingesehen am 15.01.2013]

Oracle-Dokumentation: JavaFX 2 Documentation URL: <http://docs.oracle.com/javafx/> [eingesehen am 12.04.2013]

Abbildungen

Komponenten	Swing	JavaFX
Performance	in bestimmten Bereichen eingeschränkt	durch Hardwarebeschleunigung sehr performant
Einsatz	Webanwendungen, Desktopanwendungen	
		Mobile Anwendungen
Komponenten	unterschiedliche GUI-Elemente, selbstgezeichnete Komponenten	
		Web rendering engine Media engine
Gestaltung/Design	wird im Code festgelegt	
	Layouts	Panes CSS FXML
Portabilität	kann in mehreren Betriebssystemen benutzt werden	
Implementierung	in Java integriert	
	3 Top-Level-Container: JFrame, JApplet, JDialog	ein Top-Level-Container: Stage
Handling	Komponenten werden über Events angesprochen	
	AWTEvents, ChangeEvents ...	InputEvent, ActionEvent ...
Verbreitung	großer Bekanntheitsgrad	großes Interesse
		noch nicht im Einsatz

Abbildung 1: Vergleich: Swing - JavaFX

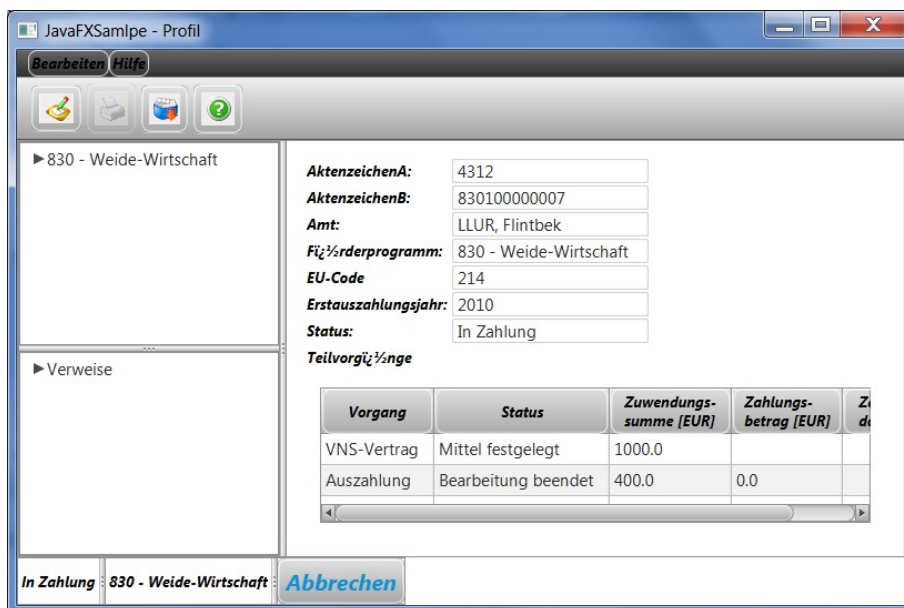


Abbildung 2: Überblick der Beispielimplementierung

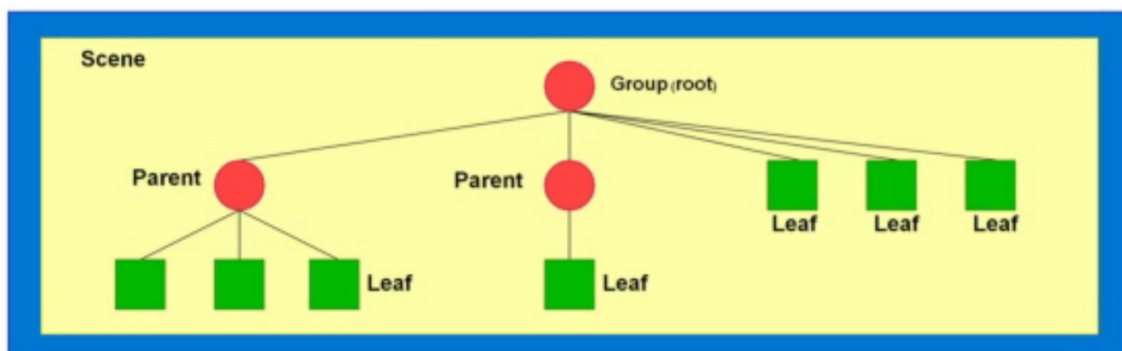


Abbildung 3: Scene - Graph