Ausarbeitung zum Logik-Builder

# Projekt im Fach IT2

## Von: Marvin Mundt, Muhammed Obi, Jan Bednarz

Idee/Einleitung

Wir haben uns bei der Auswahl unseres Projekts für einen Logikbaukasten entschieden. Die Inspiration dafür kam aus mehreren Quellen:

Dem Fach GIT (1Semester).

Der Siemens Programmiersoftware (TIA Portal).

Zum Teil auch aus Simulink.

Das TIA Portal zb dient der Programmierung von Speicher-Programmierbaren-Steuerungen (SPS) mit welchen in der Industrie Anlagen gesteuert werden. Damit nicht jeder der damit arbeitet eine Programmier-Hochsprache lernen muss gibt es die Möglichkeit das ganze als Funktionsplan, kurz FUP zu programmieren. Hierbei werden logische Bausteine verwendet, Und, Oder, SR-Flipflops usw.

Diese werden dann mit den vorher konfigurierten Eingängen beschaltet und untereinander so verbunden, wie es für die Funktion der Anlage nötig ist. Am Ende werden in Abhängigkeit dieser Schaltungen die Ausgänge der Steuerung geschaltet wodurch dann 24V Steuerspannung zb an ein Schütz angelegt werden.

Das Ganze ermöglicht eine sehr leicht verständliche und breite Anwendung, jedoch sind diese Programme allesamt sehr teuer.

Wir wollten also einen solchen Logikbaukasten entwickeln, wenn auch in der Funktion nicht ganz so umfangreich. Am Ende sollten mit dem Programm Grundlagenschaltungen simulierbar sein, also Schaltungen aus Oder- sowie Und-Bausteinen. Diese sollte man aus einem Auswahlfenster auf die “Arbeitsfläche” ziehen und dann durch Linien verbinden können. Man hätte dann sehen sollen wie die einzelnen Bausteine ihre Inputs und Outputs verarbeiten und was dabei rauskommt.

### Grafische Umsetzung/Positionierung von Bausteinen

Da wir sowohl feste als auch programmierte graphische Objekte verwenden wurde es früh nötig eine verschachtelte MVC Struktur zu nutzen.

In diesem Absatz wird die initiale (graphische)Erstellung eines Bausteinobjekts erklärt sowie die Drag/Drop Funktion.

Auf die Verschachtelte MVC Struktur wird in einem anderen Absatz separat eingegangen.

Feste Grafiken im Haupt Controller:

Alle Festen Grafiken sowie die jeweiligen Zonen sind im Scenebuilder erstellt worden. Es gibt einmal Sources , also Quellen, und das Outputarea. Außerdem gibt es noch das DeleteFeld.

Ihre Funktion wird im MainController (Controller) hergestellt. Dort sind sie auch per @FXML angebunden

Zb: @FXML  
private HBox sourceUnd;

Zu den Sources gehören sourceUnd, sourceOder und sourceLine. Jede Source bekommt ein Image zugewiesen. Es wird also ein Image erstellt

Image baustein=new Image(imagePath+images[0])

Und ein ImageView

(ImageView sourceview =new ImageView()

Welcher dann das entsprechende Image zugewiesen bekommt

sourceview.setImage(baustein);

Und dann an die jeweilige Source angebunden wird.

sourceUnd.getChildren().add(sourceview);

Die Sources haben die selbe Funktion wie ein Button, sie werden an je zwei Handles angebunden, einmal

sourceUnd.setOnMousePressed(this::handleMousePressedUnd);

Und

sourceview.setOnDragDetected(this::sourceOnDragDetectedHandler);

Hier wird also ein Observer Pattern angewandt.

Das MouseEvent wird (in diesem Beispiel) dafür genutzt den Boolean undPressed zu aktualisieren, wenn sich die Maus über der Schaltfläche für den jeweiligen Baustein befindet. Gelichzeitig gibt es aber noch ein weiteres Observer Pattern, nämlich das für den Drag. DragDetected ist ein MouseEvent, beim initialen Klick wird ein eine Methode ausgeführt. Für den Imagedrag wird nämlich ein Clipboard verwendet, dort wird beim ersten Klick das betroffene Image abgelegt, indem ein neuer ClipboardContent erzeugt wird. Dieser ClipboardContent kann von allen DragEvents verwendet werden. Damit der Baustein nur im OutputArea abgelegt werden kann, überprüft ein DragEvent, ob die GestureSource nicht der gewünschten Ablagefläche entspricht. Ist diese Bedingung erfüllt wird der TransferMode”Copy\_Or\_Move” angewandt.

Copy\_Or\_Move wird in der Dokumentation als “Conveniance Constant” beschrieben, erfüllt aber in diesem Fall die selbe Funktion wie setTranslateX/Y. Es werden also die Translates (Koordinaten) des Images bzw des ClipboardContents denen der aktuellen Mausposition angepasst.

Ist der Drag abgeschlossen, wird also die Maustaste losgelassen, so wird der Boolean der jeweiligen Source ausgewertet. Abhängig davon welcher von denen zu diesem Zeitpunkt true ist wird ein dementsprechender Baustein an den aktuellen Mauskoordinaten erstellt. Ist dies geschehen wird der Boolean wieder false, die Aktion ist abgeschlossen.

Nicht feste Grafiken&PaneView:

Die Bausteine zählen zu den nicht festen Grafiken. Damit ist gemeint, dass sie nicht von Anfang an im SceneBuilder existieren. Sie sind programmiert und können somit nicht über @FXML eingebunden werden. Sie werden in die PicturePane eingefügt, welche dem OutputArea entspricht. Alles, was über die PicturePane eingefügt wird wird im PaneView erstellt.

Soll nun ein Baustein erzeugt werden so werden die Mauskoordinaten an der jeweiligen Position erfasst und über PaneMain an das PaneModel weitergegeben. Aufgrund der dort nun vorliegenden Koordinaten für X und Y kann mit dem Aufruf von addUndGroup ein Bausteinobjekt erzeugt werden.

AddUndGroup:

AddUndGroup ist die Methode welche den Konstruktor der Und-Bausteingruppe aufruft. Der Ablauf ist bei Oder abgesehen von den Namen der gleiche und wird nicht extra beschrieben.

Bei der Grafik des Bausteines handelt es sich um eine Gruppe aus mehreren Panes. Es gibt einmal die Grundfläche und die Verbinder. Bei der Bildung der Gruppe werden 3 Verbinder auf der Grundfläche über Translates platziert.

Beispiel für den Ausgansverbinder: Translate transUR = new Translate(45, 40);  
KleinUntenRechts.getTransforms().add(transUR);

Abschließen erhält der Konstruktor vom Model die aktuellen Mauskoordinaten welche dann bestimmen an welcher Stelle des OutputAreas der Baustein erscheint.

undBaustein.setTranslateX(paneModel.gibX());  
undBaustein.setTranslateY(paneModel.gibY());

Am Ende wird die Gruppe mit den jeweiligen Koordinaten zurückgegeben.

Das Ganze ist also für jeden individuellen Baustein wiederverwendbar.

AddLine:

Das Hinzufügen von Linien unterscheidet sich nur minimal vom Hinzufügen der Bausteingrafiken. Die dafür angelegte Source verhält sich ebenfalls wie ein Button, wird jedoch nicht von dort auf die OutputArea gezogen. Stattdessen muss man die Fläche nur einmal anklicken. Durch ein MouseEvent wird der Boolean linePressed zu “true” ausgewertet. Wenn man nun auf dem OutputArea die Maus drückt , zieht, und wieder loslässt werden je die X und Y Werte des Startpunktes sowie des Endpunktes erfasst und an das Model übergeben. Abhängig von diesen Werten und davon ob der Boolean linePressed true ist, wird dann eine Linie erstellt mit welcher Bausteine verbunden werden können. Die umsetzung dieses Features ist leider nicht über das Grafische hinaus gekommen, die Funktion der Bausteine musste priorisiert werden.

Verschieben von Bausteinen&PaneController:

Damit die Schaltung die der User erstellen möchte auch angepasst werden kann ist es nötig die Bausteine auch nach der Erstellung verschieben zu können. Hierbei konnten wir nicht wie vorher ein DragEvent und ein ClipBoard verwenden, da das ClipBoard nur einige wenige Datentypen akzeptiert und Gruppen nicht dazu gehören. Außerdem werden die Objekte auf der PicturePane abgelegt, nicht direkt auf dem OutputArea. Dadurch kann das Verschieben nicht im MainController passieren sondern muss im PaneController durchgeführt werden.

Beim Weiterverschieben wird das händisch umgesetzt was zuvor durch Copy\_Or\_Move automatisch gemacht wurde. Die Pane auf der die Objekte abgelegt wurden wird an ein MouseEvent angebunden, in Abhängigkeit davon wird, wenn es sich um die primäre, also die linke Maustaste handelt, die Methode “dragger” ausgeführt.

paneView.getPicturePane().  
 setOnMousePressed(this::handleMousePressed);  
}

public void handleMousePressed(MouseEvent mouseEvent) {  
 if (mouseEvent.getButton() == MouseButton.*PRIMARY*) {  
 drag();

Hier findet also das Observer Pattern wieder Verwendung.

Die drag Methode erfasst alle Children aud der Pane und wendet auf sie die dragger Methode an. Diese besagt wiederrum, dass wenn eine Node, egal ob oder, und, oder eine Linie, einen Mausklick registriert, sie von dort an die Koordinaten der Maus als ihre eigenen verwendet, bis die Maustaste losgelassen wird.

Damit das mit den Koordinaten funktioniert müssen zwei Rechnungen durchgeführt werden. Diese, wie auch das Zwischenspeichern und Abrufen sämtlicher Werte läuft über das Model.

In der ersten Rechnung wird ermittelt wo sich das Objekt grade befindet. Dafür benötigen wir die Translates der Node(node.getTranslateX()/Y) sowie die aktuelle Mausposition (currX/Y).

ZwischenwertX=currX-posXNode;  
ZwischenwertY=currY-posYNode;

In der Zweiten Rechnung wird der neue Translate Wert für die Node errechnet. Diese wird durchgeführt, solange sich die Maus bewegt.

nodeXNeu=currX-ZwischenwertX;  
nodeYNeu=currY-ZwischenwertY;

Der neue Node Wert wird dann an die Node übergeben, der letzte Wert ist dann die neue Position.

Im Nachhinein wäre es eventuell schlauer oder schöner gewesen hierbei beim ersten Klick die Node zu löschen, an den Mauskoordinaten ein Bild von ihr mit Translates zu verschieben, und an der Letzten Position ein neues Objekt zu erzeugen. Der Weg den wir gewählt haben funktioniert zwar, jedoch ist das ganze recht ruckelig.

Da wir hier generell alle Nodes auf der Pane ansprechen kann so allerdings auch jedes Objekt bewegt werden. Es ist eine allgemeingültige Lösung.

Entfernen von Objekten:

Es kann vorkommen dass ein User einen Baustein oder eine Linie nicht benötigt. Also muss es auch möglich sein, Objekte wieder zu entfernen. Das geschieht ebenfalls mithilfe von MouseEvents.

Bei der Abfrage zum Verschieben von Objekten wird der Primary Mouse Button genutzt.

Der Secondary Button ist also noch frei. Durch diesen wird nun die Methode “deleter” aktiviert. Diese wird erstmal auf alle Nodes angewendet, jedoch wird im deleter dann nochmal ein Mausklick abgefragt, diesmal auf eine bestimmte Node. Diese wird dann mit .remove(node) entfernt. Dadurch hat man auch einen Doppelklick Effekt welcher vor leichtsinnigem oder versehentlichem Entfernen von Objekten bewahrt.