

Projekt Softwaretechnik Wintersemester 16/17

Flow Design



Felix Grammling

Matr. Nr.: 749521

fegrit00@hs-esslingen.de

Samuel Maier

Matr. Nr.: 749770

samait01@hs-esslingen.de

David Hössle

Matr. Nr.: 749809

dahoit02@hs-esslingen.de

Luca Heft

Matr. Nr.: 750142

luheit01@hs-esslingen.de

Version 1.0

17.01.2017

Inhaltsverzeichnis

Inhaltsverzeichnis	2
Abbildungsverzeichnis.....	4
Tabellenverzeichnis.....	4
Einleitung.....	5
1.1 Das Entwicklerteam.....	5
1.2 IT-Designers Gruppe	5
1.3 Das Projekt	6
2 Projektmanagement	7
2.1 Projektmanagementmethode	7
2.2 Versionskontrolle.....	8
2.3 Treffen	9
2.3.1 Treffen im Team.....	9
2.3.2 Treffen mit dem Betreuer	9
2.3.3 Treffen mit dem Kunden	9
3 Konzeption	10
3.1 Anforderungen.....	10
3.1.1 Darstellung der Diagrammarten	10
3.1.2 Einhaltung von Notationen.....	10
3.1.3 Digitale Dateiablage	10
3.1.4 Versionierung	10
3.1.5 Unterstützung bei der Eingabe	11
3.1.6 Allgemeine Anforderungen	11
3.1.7 Nicht umgesetzte Anforderungen.....	11
3.2 Auseinandersetzung mit Flow Design.....	12
3.3 Mockups	16
3.3.1 Start Dialog Fenster	16
3.3.2 Hauptfenster.....	18
4 Entwicklung	20
4.1 Aufbau der Software.....	20
4.1.1 Modul – Daten	20
4.1.2 Modul – Backend.....	20
4.1.3 Modul – Frontend	20
4.2 Datenstruktur	20

4.2.1 XML	20
4.2.2 Ordnerstruktur	25
4.2.3 Java Klassen	27
4.2.4 Enums	29
4.2.5 Testing	30
4.3 Backend	30
4.3.1 Datenspeicher	30
4.3.2 XML Parser	31
4.3.3 Schnittstellen	32
4.4 Frontend.....	33
4.4.1 Zeichenfläche.....	33
4.4.2 FileTree.....	36
4.4.3 Statusbar	38
4.4.4 Menüleiste	39
4.4.5 Komponenten und Eigenschaften.....	39
4.5 Projekt bauen	42
4.5.1 Maven	43
4.5.2 Language Level und Java Compiler	43
5 Benutzerhandbuch	44
5.1.1 Oberfläche	44
6 Ausblick.....	49
6.1 Feature Vorschläge	49
6.1.1 Rückgängig.....	49
6.1.2 Raster in der Zeichenfläche	49
6.1.3 Zeichenfläche immer verschiebbar.....	50
6.1.4 Zum Mauszeiger zoomen	50
6.1.5 Beim Erstellen eines Diagramms den Parent angeben	50
6.1.6 Gruppierung.....	50
6.1.7 Code Generierung.....	50
7 Fazit.....	51
7.1 Herausforderungen	51
7.2 Erkenntnisse.....	51
7.3 Danksagungen	52
Literaturverzeichnis	53

Abbildungsverzeichnis

Abbildung 1 - Trello: Kanban Board	7
Abbildung 2 - Tafelaufschrieb zum Thema Flow Design	12
Abbildung 3 - Tafelaufschrieb zum Thema Flow Design	13
Abbildung 4 - Tafelaufschrieb zum Thema Flow Design	13
Abbildung 5 - Tafelaufschrieb zum Thema Flow Design	14
Abbildung 6 – Split: altes Konzept nach Ralf Westphal	14
Abbildung 7 - Split: neues Konzept nach Stefan Lieser	15
Abbildung 8 - Mockup des Start Dialog Fensters.....	16
Abbildung 9 - Mockup des Start Dialog Fensters: Neues Projekt erstellen	17
Abbildung 10 - Mockup des Hauptfensters	18
Abbildung 11 - XML Datenstruktur der Projektdatei.....	21
Abbildung 12 - XML Datenstruktur der Diagramme	22
Abbildung 13 - Ausschnitt aus der XML Datenstruktur eines Diagramms.....	22
Abbildung 14 - Ausschnitt aus der XML Datenstruktur eines Diagramms.....	23
Abbildung 15 - Ausschnitt aus der XML Datenstruktur eines Diagramms.....	23
Abbildung 16 - Ausschnitt aus der XML Datenstruktur eines Diagramms.....	24
Abbildung 17 - Ausschnitt aus der XML Datenstruktur eines Diagramms.....	24
Abbildung 18 - Ausschnitt aus der XML Datenstruktur eines Diagramms.....	25
Abbildung 19 - Ordnerstruktur eines Projekts	26
Abbildung 20 - Java Class Diagramm 1	27
Abbildung 21 - Java Class Diagramm 2	28
Abbildung 22 – Treeview	36
Abbildung 23 – Statusbar.....	38
Abbildung 24 - Menüleiste	39
Abbildung 25 - Komponenten: Basic Forms	40
Abbildung 26 – Eigenschaften eines Diagrammes	41
Abbildung 27 - Eigenschaften einer Node.....	42
Abbildung 28 - Oberfläche: Start Dialog Fenster	45
Abbildung 29 - Oberfläche: Filetree	45
Abbildung 30 - Oberfläche: Zeichenfläche.....	46
Abbildung 31 - Oberfläche: Eigenschaften und Komponenten.....	47
Abbildung 32 - Oberfläche: Menüleiste	48
Abbildung 33 - Raster der Unreal Engine 4.....	49
Abbildung 34 - Gruppierung von Elementen in einem Diagramm.....	50

Tabellenverzeichnis

Tabelle 1 - Verbindungspunkte.....	34
------------------------------------	----

Einleitung

Diese Software ist als Projektarbeit im Bereich Softwaretechnik an der Hochschule Esslingen in Kooperation mit der IT-Designers Gruppe entstanden. Der Kunde unserer Software war hierbei Herr Kevin Erath von der IT-Designers Gruppe. Zu Anfang erhielten wir Betreuung bezüglich technischer und fachlicher Fragen von Prof. Dr.-Ing. Andreas Rößler von der Hochschule Esslingen, jedoch änderte sich dies wegen kurzfristigen Ereignissen, sodass wir von Dipl.-Ing.(FH), MSc. Ömer Haybat, welcher ebenfalls bei der IT-Designers Gruppe angestellt ist, betreut wurden.

1.1 Das Entwicklerteam

Das Entwicklerteam setzt sich aus vier Mitgliedern zusammen:

Luca Heft

David Hössle

Samuel Maier

Felix Grammling

Anders als üblich, entschlossen wir uns bereits zu Beginn des Projekts keine klassische Rollenverteilung innerhalb unseres Teams zu übernehmen. Da keiner von uns eine Spezialisierung aufweisen konnte, hatte somit jeder von uns die Möglichkeit Einblicke in jeden Bereich der verschiedenen Rollen zu erhalten. Des Weiteren war es uns so einfacher möglich den Ausfall eines Teammitglieds, sei es durch Krankheit oder zeitweise Unerreichbarkeit, zu kompensieren.

1.2 IT-Designers Gruppe¹

Die IT-Designers Gruppe ist ein Verbund aus den Unternehmen IT-Designers GmbH und Steinbeis Transferzentrum Softwaretechnik. Als ein mittelständisches Unternehmen, welches im Großraum Stuttgart angesiedelt ist, bieten Sie für Ihre Kunden individuelle Softwarelösungen an. Auf Grund guter Beziehungen zur Fachhochschule Esslingen, haben sie bereits häufiger Studienprojekte vorgeschlagen und betreut.

¹ [online] <http://www.it-designers-gruppe.de/it-designers-gruppe/>

1.3 Das Projekt

Das Schreiben von Programmcode lässt sich, in den meisten Fällen, schnell und einfach erlernen. Die größte Herausforderung dabei ist jedoch, den Code übersichtlich, effektiv, und unabhängig zu halten. Das Konzept Flow Design, welches von Ralf Westphal und Stefan Lieser entwickelt wurde, soll den Softwareentwickler dabei unterstützen. Zusätzlich bietet das Konzept die Möglichkeit, den Kunden in den Entwicklungsprozess mit einzubeziehen, um die Software ganz nach seinen Wünschen zu erstellen.

Das Konzept baut dabei auf drei Diagrammarten auf:

- **Das System Umwelt Diagramm**

In diesem Diagramm werden Akteuren und Ressourcen in Abhängigkeit des Systems dargestellt. Es kann mit dem klassischen Kontextdiagramm verglichen werden. Es hilft bei großen Projekten den Überblick zu halten, sodass diese nicht vom ursprünglichen Plan abweichen.

- **Das Dialog Diagramm**

In diesem Diagramm können erste Designideen ausgedacht und dargestellt werden. So kann dem Kunden ein erster Eindruck über die spätere Software gegeben und auf Wünsche seinerseits eingegangen werden. Des Weiteren können mögliche Interaktionen, den sogenannten „Delegationen“, der Anwendung festgelegt werden. Zum Beispiel das Drücken eines Buttons.

- **Das Flow Design Diagramm**

Das Flow Design Diagramm ist das Herzstück dieses Entwurfskonzepts. Es beschreibt detailliert den Programmfluss einer solchen Interaktion. Über Funktionseinheiten, welche eine minimalistische Funktion beschreiben, wird die Verarbeitung des Datenflusses dargestellt. Der Datenfluss baut dazu auf das sogenannte EVA-Prinzip auf. Jede Funktionseinheit definiert einen Eingang, in dem die Daten empfangen werden, eine Funktionseinheit, die diese Daten verarbeitet, und einen Ausgang, in dem die verarbeiteten Daten wieder ausgegeben werden. Dabei können sich die Eingänge und Ausgänge an Ihrer Anzahl unterscheiden. Um den Datenfluss besser zu verdeutlichen, soll dieser hierbei nur in eine Richtung verlaufen.

Stellt eine Funktionseinheit eine zu komplexe Operation bereit, kann diese als ein eigenes Flow Design Diagramm dargestellt werden, welches ebenfalls auf das obige Prinzip aufbaut.

2 Projektmanagement

Jedes Projekt muss geplant werden. Dabei ist das Projektmanagement ein wichtiger Punkt. In diesem Abschnitt erläutern wir, wie wir unser Projekt aus dieser Sicht umgesetzt haben und welche Systeme dabei eine Rolle spielten.

2.1 Projektmanagementmethode

Da unser Team nur aus wenigen Mitglieder bestand, und uns die agile Projektmanagementmethode SCRUM zu „übertrieben“ für dieses Projekt schien, haben wir uns für die Projektmanagementmethode „Kanban“ entschieden.

Kanban baut auf den sogenannten „Boards“ auf, die jeweils einen selbst definierten Zustand beschreiben. Um den Entwicklungsprozess zu unterstützen und übersichtlicher zu gestalten können nun sogenannte „Karteikarten“, die die eigentliche Aufgabe enthalten, in den Boards abgelegt werden. Je nach Zustand einer Karteikarte liegt diese in dem entsprechenden Board.

Dazu verwendeten wir die web-basierte Projekt-Management-Software „Trello“.

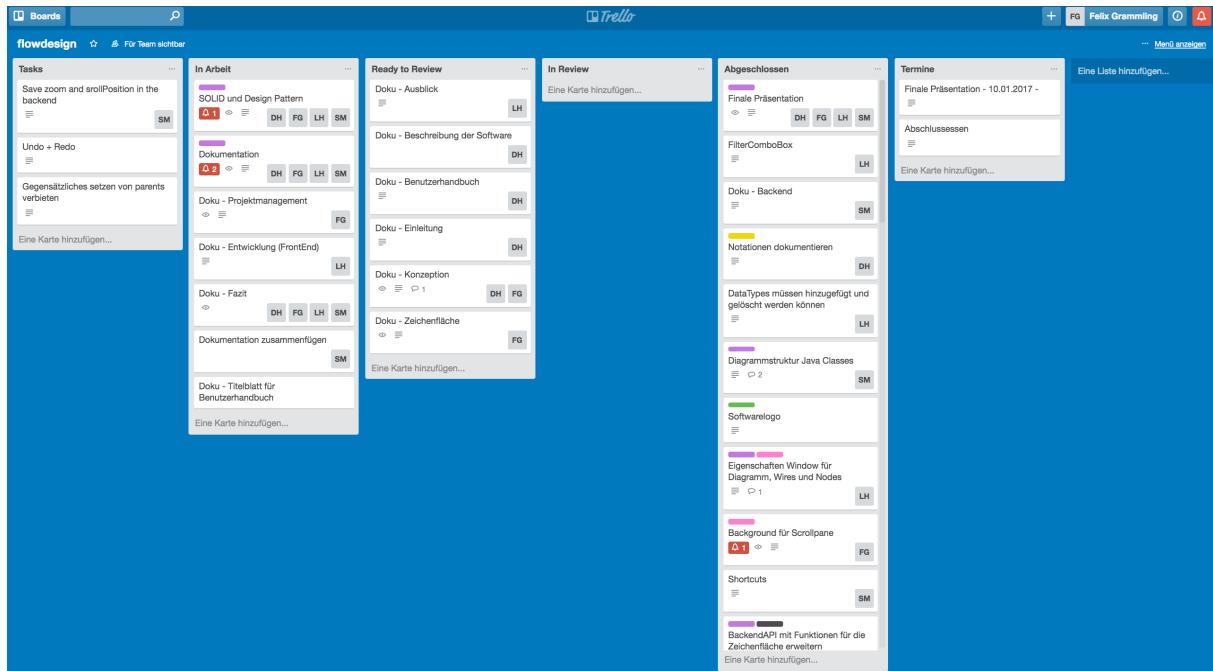


Abbildung 1 - Trello: Kanban Board

Unser Kanban-Board hatte folgende Struktur:

- **Tasks**

In das Board „Tasks“ haben wir alle Aufgaben die während unseres Projektes aufgetreten sind erstellt und abgelegt.

- **In Arbeit**

In das Board „In Arbeit“ haben wir alle Aufgaben abgelegt, die von einem Teammitglied aktiv bearbeitet werden. So sieht jedes Teammitglied, welche Aufgabe zurzeit von wem bearbeitet wird.

- **Ready to Review**

In das Board „Ready to Review“ haben wir alle Aufgaben abgelegt, die von einem Teammitglied fertig bearbeitet wurden. Andere Teammitglieder sehen nun, dass diese Aufgabe bereit zur Kontrolle ist.

- **In Review**

In das Board „In Review“ haben wir alle Aufgaben abgelegt, die aktiv von einem Teammitglied zur Kontrolle eingesehen werden. Lief die Kontrolle erfolgreich, kann die Aufgabe in das Board „Abgeschlossen“ verschoben werden. Lief die Kontrolle nicht erfolgreich wird die Aufgabe wieder in das Board „In Arbeit“ gelegt.

- **Abgeschlossen**

In das Board „Abgeschlossen“ haben wir alle Aufgaben abgelegt, die erfolgreich die Kontrolle durchlaufen haben und damit abgeschlossen wurden.

- **Termine**

In das Board „Termine“ haben wir wichtige Termine abgelegt, wie zum Beispiel die Treffen mit unserem Kunden oder Betreuer.

2.2 Versionskontrolle

In der Softwareentwicklung spielt die Versionsverwaltung eine wichtige Rolle. Umso mehr, wenn in einem Team entwickelt wird. Sie hilft dem Team die Kontrolle über den Code zu halten, dass bedeutet es können Änderungen leichter nachvollzogen und eventuelle Fehler leichter rückgängig gemacht werden.

Die Versionsverwaltung haben wir über den web-basierten Versionsverwaltungsdienst „**Bitbucket**“ von der Firma „Atlassian“ realisiert. Als Versionsverwaltungssystem kam dabei „**Git**“ zum Einsatz.

Damit stets eine lauffähige Version unserer Anwendung bereitgestellt werden konnte, haben wir unseren Entwicklungsprozess in zwei Versionen (Repositories) aufgeteilt.

Die eben erwähnte immer lauffähige Version wurde in unserem „Haupt“-Repository umgesetzt. So konnte zu jedem Zeitpunkt sichergestellt werden, dass der Entwicklungsprozess anderer Teammitglieder nicht beeinträchtigt wurde.

Das zweite Repository, unser „Test“- Repository, wurde wie der Name schon erschließen lässt, zum Testen von Funktionen verwendet.

2.3 Treffen

2.3.1 Treffen im Team

Uns war es im Team ein Anliegen jeden einzelnen im Entwicklungsprozess mit einzubeziehen, um den Lernerfolg jedes Teammitglieds so hoch wie möglich zu gestalten. Aus diesem Grund war jeder im Team bei der Entwicklung beteiligt. Deshalb war es umso notwendiger regelmäßige Treffen zu Veranstalten. Da wir uns im Team bereits aus früheren Studienprojekten kannten, stellte das für uns keine große Herausforderung dar. Durch ähnliche Stundenpläne konnten wir uns mindestens jede Woche zweimal Treffen um die aktuelle Situation zu besprechen und uns gegenseitig zu unterstützen.

2.3.2 Treffen mit dem Betreuer

Wie bereits erwähnt übernahm Herr Haybat von der IT-Designers Gruppe die Betreuung unseres Teams. Nach Absprache mit Ihm, einigten wir uns auf ein Treffen alle zwei Wochen. Dort konnten wir Ihm unseren aktuellen Stand der Anwendung zeigen und anstehende Themen diskutieren. Er gab uns Hilfestellung zu allen möglichen Problemen und lehrte uns Tipps für die Entwicklung von „sauberer“ Software. Unteranderem ist hier das SOLID-Prinzip und allerlei Entwurfsmuster in der Softwareentwicklung zu erwähnen.

2.3.3 Treffen mit dem Kunden

Durch eine gute Vorbereitung unseres Projektthemas, durch eigene Recherche und natürlich durch den Vorteil auf ein bereits vorhandenes Studienprojekt zum Thema „Flow Design“ zurückgreifen zu können, war uns schnell bewusst, welche Anforderung an uns von unserem Kunden Herr Erath, ebenfalls von der IT-Designers Gruppe, gestellt werden.

Im Laufe des Projekts hatten wir leider nur zweimal die Möglichkeit genutzt um mit dem Kunden in Kontakt zu treten. Das lag leider auch an einem Irrtum unsererseits, indem wir glaubten bereits über unserem Betreuer den Kundenkontakt zu pflegen.

In den beiden Treffen nutzen wir aber die Gelegenheit um den Zwischenstand unseres Produkts vorzustellen und Verbesserungsvorschläge entgegenzunehmen.

3 Konzeption

3.1 Anforderungen

Vom Kunden wurden folgende Anforderungen gestellt:

3.1.1 Darstellung der Diagrammarten

- Die Software muss folgende drei Diagrammtypen darstellen:
 - o System Umwelt Diagramm:
Die Software muss dem Anwender die Möglichkeit bieten, über ein System Umwelt Diagramm die Systemgrenzen darzustellen.
 - o Dialog Diagramm:
Die Software muss dem Anwender die Möglichkeit bieten, über ein Dialog Diagramm Anwendungsfälle anhand einfacher Dialogentwürfe detailliert darzustellen.
 - o Flow Design:
Die Software muss dem Anwender die Möglichkeit bieten, über ein Flow Design Diagramm Interaktionen zu beschreiben.

3.1.2 Einhaltung von Notationen

- Die Software muss den Anwender bei der Einhaltung von Notationen unterstützen.

3.1.3 Digitale Dateiablage

- Die Software muss dem Anwender die Möglichkeit bieten, neue Projekte anzulegen.
- Der Nutzer muss die Möglichkeit haben, das erstellte Projekte, eingeschlossen alle Diagramme, als Dateien abzuspeichern.
- Die Software muss dem Anwender die Möglichkeit bieten, eine beliebige Anzahl an Diagrammen zu erstellen.
- Die vom Anwender erstellten Projektdaten müssen einfach austauschbar sein (zum Beispiel per Mail, Fileshares).

3.1.4 Versionierung

- Die Software muss dem Anwender die Möglichkeit bieten, Versionen eines Diagramms anzulegen.

3.1.5 Unterstützung bei der Eingabe

- Der User muss bei der Eingabe unterstützt werden.
- Die Software soll den User bei der Ausrichtung von Elementen unterstützen.
- Dem Nutzer muss es möglich sein, Verlinkungen zwischen den Diagrammen zu erstellen.
- Die Software soll vom Anwender intuitiv benutzbar sein.
- Der Anwender soll sich auf bestimmte Teile der Software fokussieren können.

3.1.6 Allgemeine Anforderungen

- Die Software muss auf dem Betriebssystem Windows 7 und höher verwendbar sein.

3.1.7 Nicht umgesetzte Anforderungen

- Dem User soll es möglich sein Code aus Diagrammen heraus zu generieren.
- Die Software soll anhand von Code, Diagramme nach dem Konzept von Flow Design generieren.

3.2 Auseinandersetzung mit Flow Design

Nachdem wir das Thema Flow Design erhielten, stellten wir schnell fest, dass es sich als zeitraubend herausstellte, Informationen über dieses Thema zu erarbeiten. Aus diesem Grund suchten wir zunächst Quellen zu diesem Thema, um diese dann zu studieren. Um einen ersten Eindruck zu diesem Thema zu erlangen, stellten sich folgende Medien als hilfreich heraus:

- Ein YouTube-Video, in dem Ralph Westphal das Konzept FlowDesign präsentiert.²
- Ein Blog-Eintrag von Ralph-Westphal bei GeeksWithBlogs.net³
- Das Buch von Ralph-Westphal „The Architect's Napkin – Der Schummelzettel“⁴

Anhand dieser Quellen setzen wir uns beim nächsten Gruppentreffen zusammen, um jedem im Team das Konzept verständlich zu machen. Folgende Bilder zeigen unsere ersten Entwurfskonzepte zum Thema Flow Design.

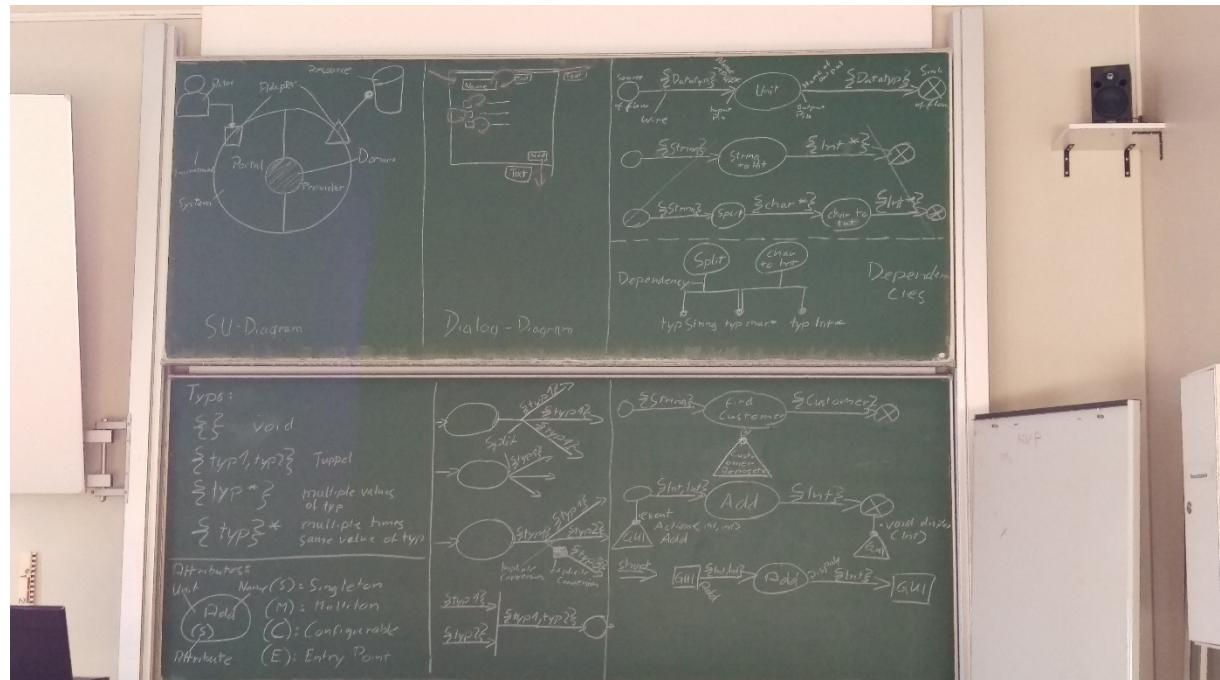


Abbildung 2 - Tafelaufschrieb zum Thema Flow Design

²[online] <https://www.youtube.com/watch?v=I36OImYMIVs>

³[online] <http://geekswithblogs.net/theArchitectsNapkin/archive/2011/03/19/flow-design-cheat-sheet-ndash-part-i-notation.aspx>

⁴Ralf Westphal, [online] <https://leanpub.com/thearchitectsnapkin-derschummelzettel>

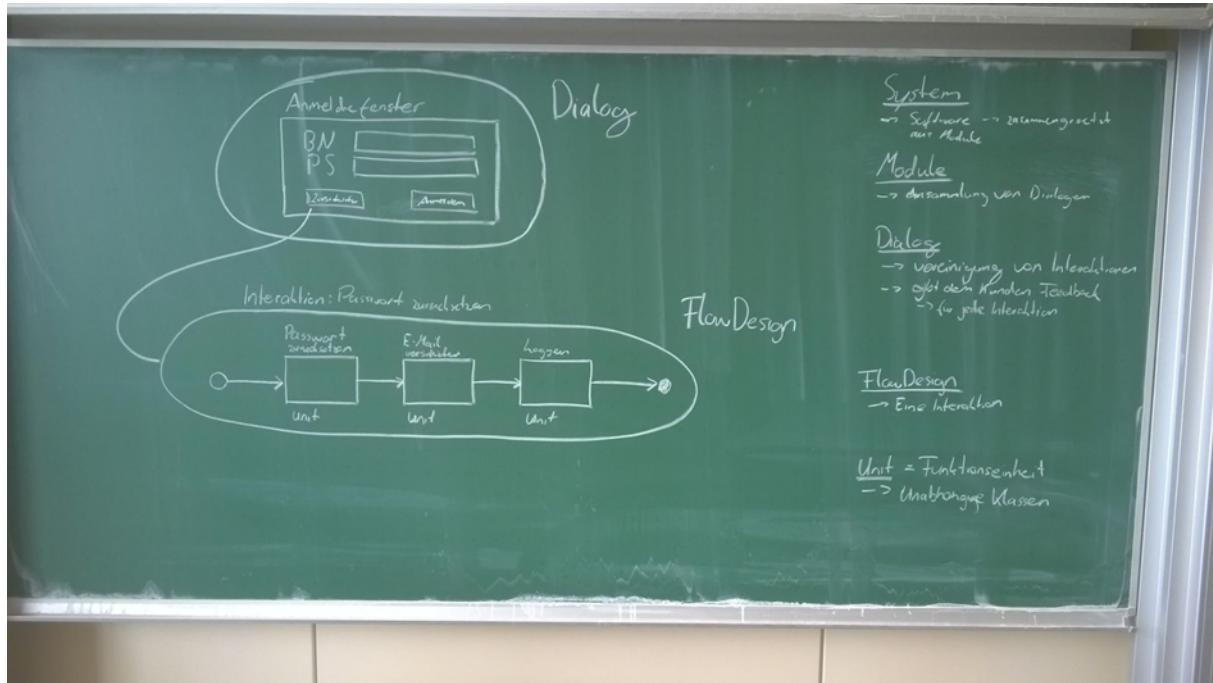


Abbildung 3 - Tafelaufschrieb zum Thema Flow Design

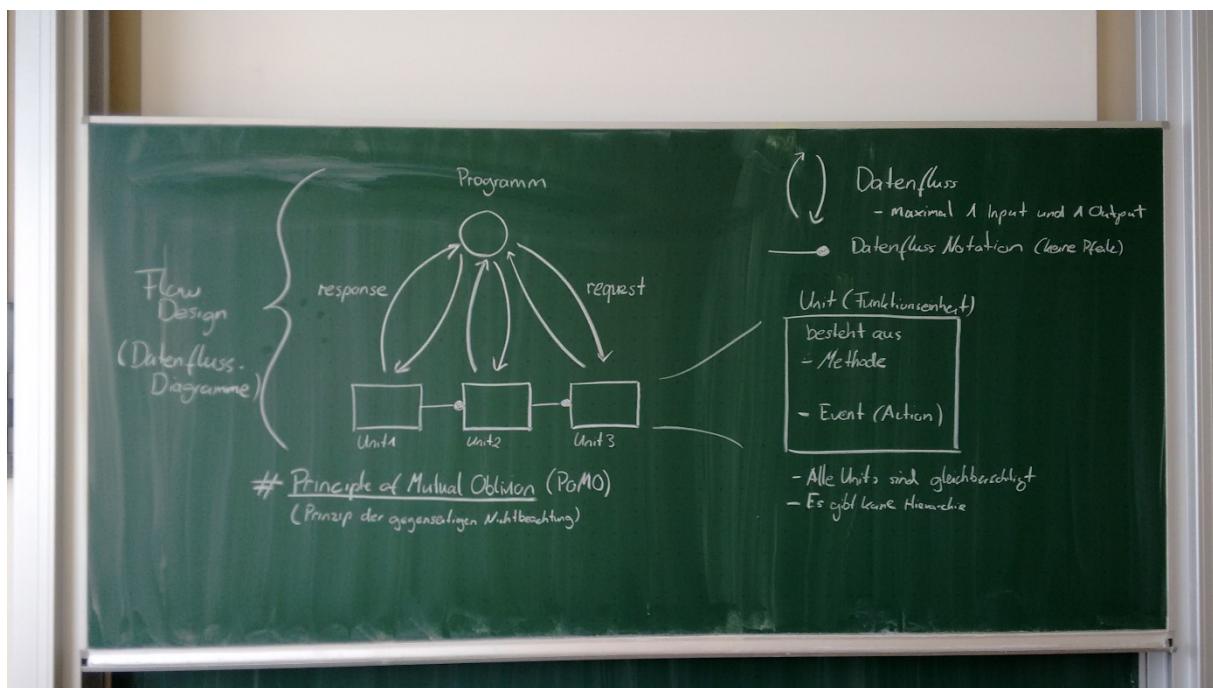


Abbildung 4 - Tafelaufschrieb zum Thema Flow Design

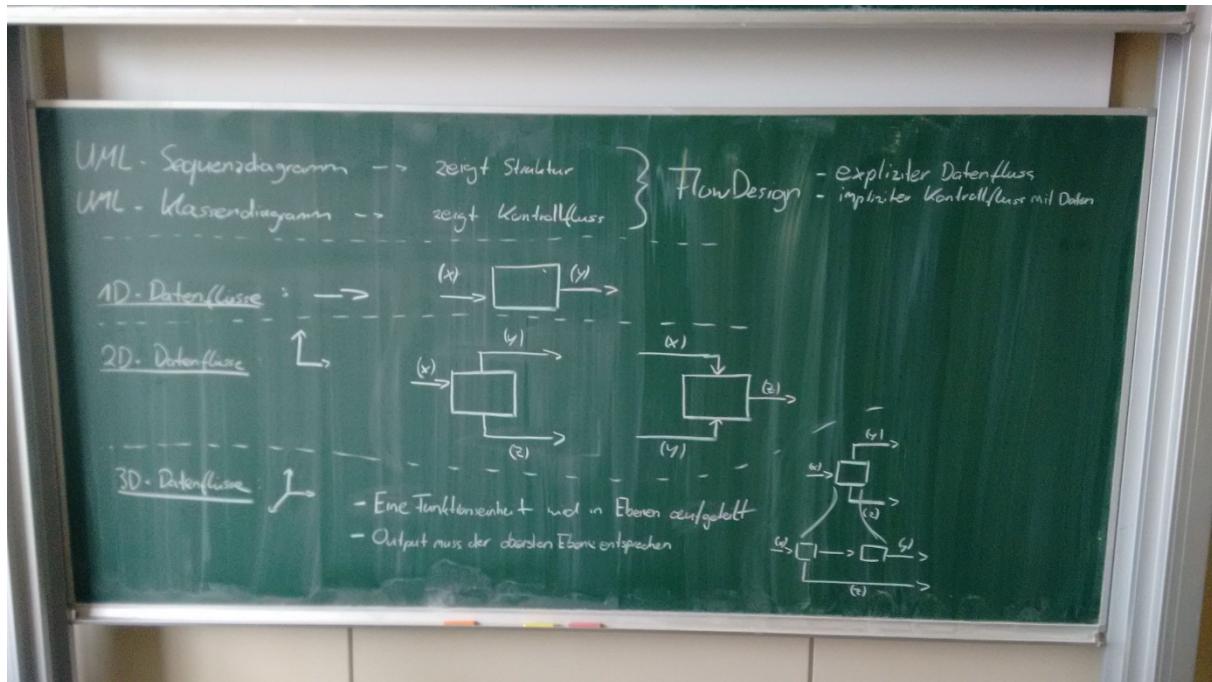


Abbildung 5 - Tafelaufschrieb zum Thema Flow Design

Nachdem wir uns in das Thema eingearbeitet hatten, erhielten wir von unserem Betreuer ein Dokument⁵, welches nur wenige Tage zuvor, von Stefan Lieser, der andere Mitbegründer von Flow Design, erstellt wurde. Wir versuchten dieses neue Wissen in unsere Anwendung einzubauen. Jedoch ist unsere Software in einigen Punkten immer noch an das ursprüngliche Konzept von Ralf Westphal, aus dem Jahre 2011, angelehnt.

Einer der wohl wesentlichsten Unterschiede dürfte hierbei die Funktionsweise der Funktionsseinheit „Split“ sein. Die ursprüngliche Split-Einheit teilte den Datenstrom (x, y) in beliebig viele Datenströme des Typs (x, y) auf. Diese Funktionsweise wurde auch in unserer Anwendung umgesetzt.

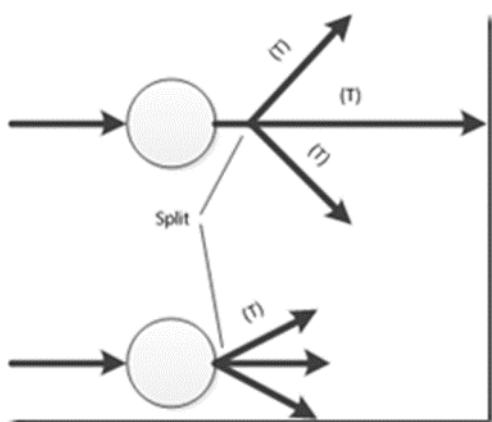


Abbildung 6 – Split: altes Konzept nach Ralf Westphal

⁵ [online] <http://refactoring-legacy-code.net/wp-content/uploads/2016/10/CheatSheet-Flow-Design.pdf>

In der neueren Version von Flow Design wird der Split verwendet um aus einem Datenstrom **(x, y)** einen Datenstrom des Typs **(x)** und einen Datenstrom des Typs **(y)** zu teilen.

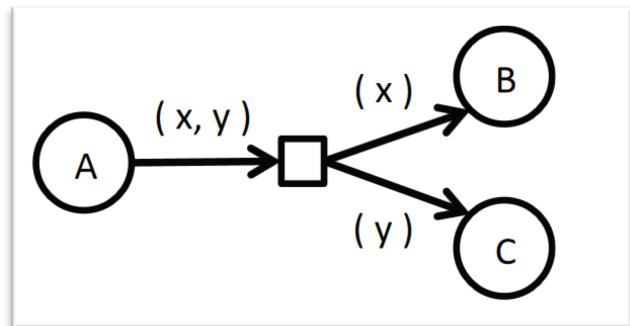


Abbildung 7 - Split: neues Konzept nach Stefan Lieser

3.3 Mockups

Schon zu Beginn des Entwicklungsprozesses haben wir uns bei der Gestaltung und Bedienung unserer Anwendung an andere Programme, wie Microsoft Visio, IntelliJ von Jetbrains oder Balsamiq, orientiert.

Letzteres verwendeten außerdem dazu, erste Mockups von unserer Anwendung zu erstellen:

3.3.1 Start Dialog Fenster

Aus IntelliJ übernahmen wir die Idee eines Start Dialog Fensters, welches direkt nach Programmstart geöffnet wird.

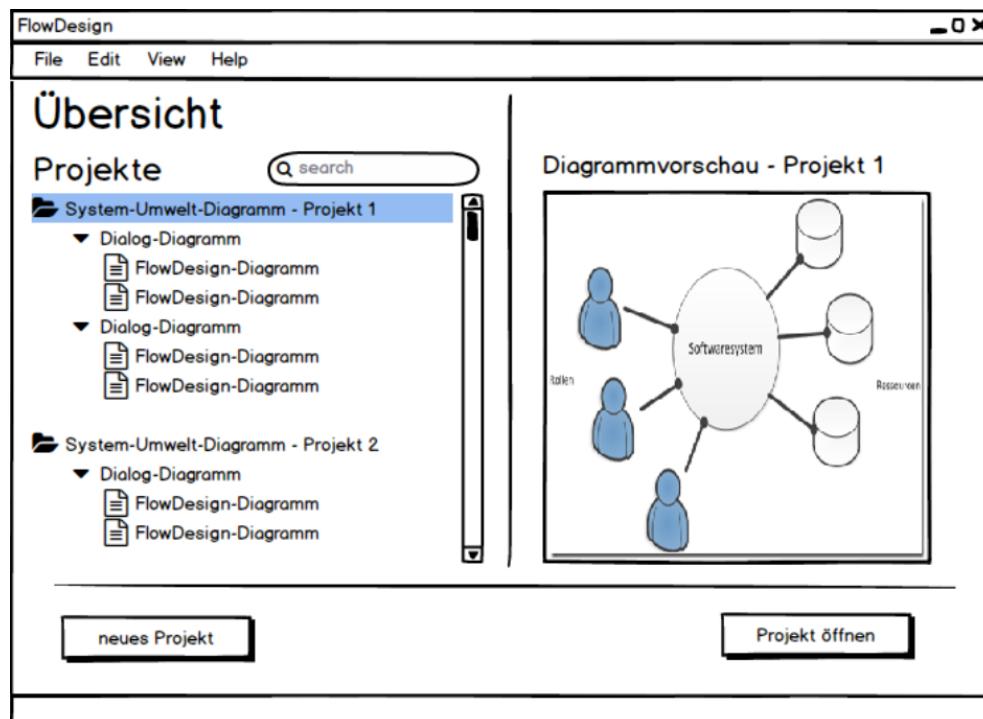


Abbildung 8 - Mockup des Start Dialog Fensters

Unser erstes Konzept sah vor, auf der linken Seite die zuletzt geöffneten Projekte und dessen Diagramme anzuzeigen. Wird ein bestimmtes Diagramm ausgewählt, würde auf der rechten Seite ein Vorschaubild dieses Diagramms dargestellt.

Nachdem wir aber feststellten, dass das Anzeigen aller Diagramme zu viel Platz in Anspruch nehmen würde, reduzierten wir die Auswahl der linken Seite auf die Projektnamen mit zugehörigen Projektpfad. Aus diesem Grund war das Anzeigen eines Vorschaubildes in dem neuen Konzept nicht nötig bzw. umsetzbar.

Wird ein neues Projekt erstellt, startet ein weiteres Dialogfenster, um dem Anwender bei der Erstellung zu unterstützen. Über diesen Dialog ist es möglich einen Projektnamen und einen Projektpfad zu definieren. Des Weiteren lassen sich direkt benötigte Diagramme durch eine einfache Checkbox-Auswahl hinzufügen.

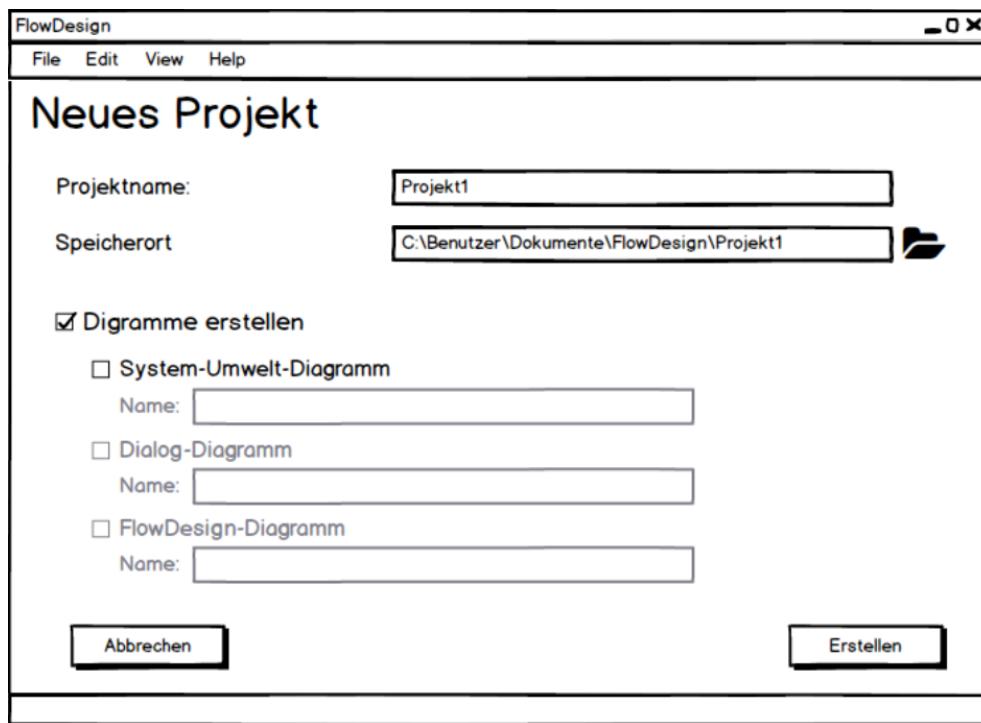


Abbildung 9 - Mockup des Start Dialog Fensters: Neues Projekt erstellen

3.3.2 Hauptfenster

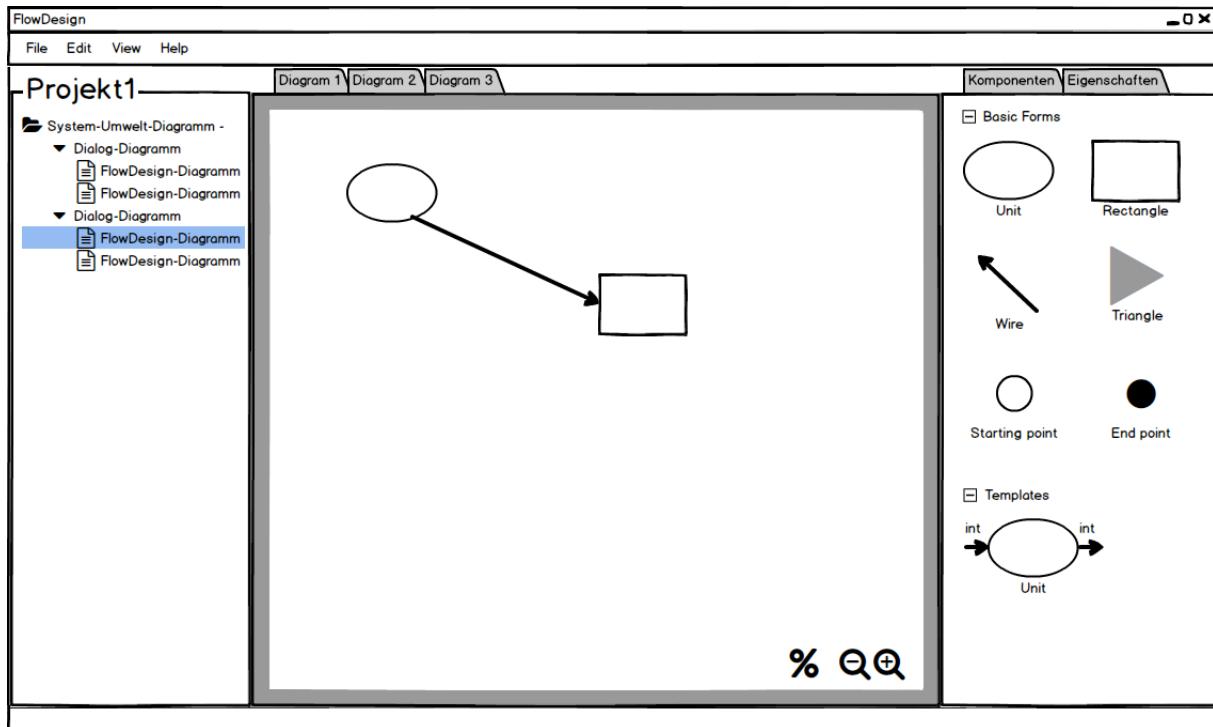


Abbildung 10 - Mockup des Hauptfensters

Unser Hauptfenster-Entwurf lässt sich grob in folgende Bereiche einteilen:

3.3.2.1 Menü

Über ein Menü soll Anwender in der Lage sein, neue Projekte und Diagramme zu erstellen. Des Weiteren sollen unterschiedlichste Einstellungen möglich sein.

3.3.2.2 Verzeichnisbaum

Im Verzeichnisbaum („FileTree“), werden dem Anwender alle vorhandenen Diagramme des geöffneten Projekts angezeigt. In der späteren Umsetzung der Anwendung, hat sich hier allerdings nur die hierarchische Darstellung geändert.

3.3.2.3 Zeichenfläche

Die Zeichenfläche, in der Mitte, ist das Herzstück der Anwendung. Hier werden die Diagramme angezeigt. Über Tabs, oberhalb der Zeichenfläche, kann der Anwender zwischen mehreren Diagrammen wechseln.

Die ursprünglich geplanten Symbole, unten rechts auf der Zeichenfläche, um die Zoomstärke anzupassen, sind entfallen. Sie würden den Anwender nur unnötig von der Kernfunktion, dem Erstellen von Diagrammen, ablenken. Die beiden Symbole, die Zoomstufe zu verringern oder zu vergrößern, sind komplett durch Drehen des Mausrades ersetzt worden. Das „%“-Symbol,

welches die Zoomstufe auf 100% stellen sollte, wurde stattdessen über einen zusätzlichen Menüeintrag gelöst.

3.3.2.4 Komponenten und Eigenschaften

Im rechten Bereich der Anwendung befindet sich eine Auswahl an allen Elementen, die der Anwender für die Modellierung des Diagramms benötigt. Je nach Diagrammart, ändern sich die zugehörigen Elemente. Über Tabs lässt sich zwischen den Eigenschaften, eines Diagramms oder Elements, und den Komponenten wechseln.

4 Entwicklung

Die Software ist in Java programmiert. Im Frontend wurde JavaFX eingesetzt. Als Entwicklungsumgebung haben wir IntelliJ von JetBrains verwendet.

4.1 Aufbau der Software

Die Software ist in drei Module aufgeteilt.

4.1.1 Modul – Daten

Im Modul Daten befinden sich alle Java Klassen der Datenstruktur. Hier gibt es aber keinerlei Funktionalität.

4.1.2 Modul – Backend

Im Modul Backend befindet sich ein Großteil der Logik des Programms. Das Backend stellt eine API zur Verfügung, über die man mit den Daten arbeiten kann.

4.1.3 Modul – Frontend

Das Modul Frontend kümmert sich um die Oberfläche. Hier befindet sich unter anderem die Zeichenfläche. Das Modul Frontend greift über die API im Backend auf die Daten zu.

4.2 Datenstruktur

4.2.1 XML

Eine der Anforderungen war, dass man ein Projekt auf der Festplatte speichern und von dort auch wieder einlesen kann. Bevor wir also mit dem Programmieren anfangen konnten, mussten wir uns Gedanken über eine sinnvolle Datenstruktur machen.

Wir entschieden uns dafür, die Daten in XML-Dateien abzuspeichern.

Wichtig: Folgende Datenstrukturen sind in der Software eins zu eins in Java Klassen umgesetzt. Für das Parsen der XML-Datenstruktur auf die Java Klassen wurde ein XML-Parser geschrieben, welcher sich in der Software im Backend zu finden ist. Beim Erweitern der Datenstruktur muss darauf geachtet werden, dass die sowohl die Java Klassen als auch der XML Parser angepasst wird!

4.2.1.1 Projektdatei

Eine weitere Anforderung war, dass jedes Diagramm in einer extra Datei abgespeichert werden soll, damit man einzelne Diagramm zum Beispiel per E-Mail verschicken kann. Da es aber auch Projektübergreifende Informationen gibt, welche abgespeichert werden müssen, wird für jedes Projekt eine XML-Datei erstellt mit der Dateiendung **flowproject**. Im nachfolgender Abbildung ist die Struktur dieser Projektdatei zu sehen.

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="ProjektName" created="DDMMJJJJ" lastedited="DDMMJJJJ">
    <diagrams>
        <diagram name="DiagramName" />
        ...
    </diagrams>
</project>
```

Abbildung 11 - XML Datenstruktur der Projektdatei

In der Projektdatei werden der Projektname, das Erstellungsdatum und das Datum der letzten Änderung gespeichert.

Des Weiteren wird in der Projektdatei der Name von jedem Diagramm gespeichert, welches zu dem Projekt dazugehört. Über den Namen des Diagramms ist es möglich, das Diagramm eindeutig zu bestimmen. So muss am Anfang lediglich diese Projektdatei in der Software ausgewählt werden und alle Diagramme können anschließend automatisch durch den Diagrammnamen importiert werden.

4.2.1.2 Diagramm-Datei

Jedes Diagramm wird in einer separaten Datei auf der Festplatte gespeichert. Die Ausarbeitung der hierfür notwendigen Datenstruktur hat einiges an Zeit beansprucht. Grund dafür war die Vielzahl an unterschiedlichen Daten, die alle in derselben Struktur abgespeichert werden sollten.

Das Entwurfskonzept Flow Design besteht aus drei verschiedene Diagrammarten. Jedes Diagramm enthält eine Vielzahl an unterschiedlicher Elemente (Funktionseinheiten, Ressourcen, etc.) welche durch Pfeile miteinander verbunden sind.

Darüber hinaus sollte die Datenstruktur in der Lage sein, mehrere Versionen des Diagramms abzuspeichern, um so zu einem späteren Zeitpunkt auf ältere Versionen zugreifen zu können.

Im Folgenden wird unsere finale Datenstruktur im Detail erläutert.

```

<?xml version="1.0" encoding="UTF-8"?>
<diagramm name="DiagramName" type="FlowDesign" parent="root" scrollPositionX="500.0" scrollPositionY="300.0" activeVersionNum="1">
    <versions>
        <version num="1" created="DDMMJJJJ" edited="DDMMJJJJ">
            <nodes>
                <node id="1" type="nodeType" x="12.9" y="36.0" width="120.4" height="65.2">
                    <text>Text</text>
                    <label>Label</label>
                    <attribut>S</attribut>
                    <link>Link</link>
                    <image>URL</image>
                    <color>blue</color>
                    <additional>Text</additional>
                    <connectors>
                        <connector id="1" orientation="connectorOrientation" type="string" />
                        ...
                    </connectors>
                </node>
                ...
            </nodes>
            <wires>
                <wire id="1" type="wireType" srcNode="nodeID" srcConn="connectionID" targetNode="nodeID" targetConn="connectionID">
                    <label>Text</label>
                    <joints>
                        <joint x="12.3" y="32.1" />
                        ...
                    </joints>
                </wire>
                ...
            </wires>
        </version>
        ...
    </versions>
</diagramm>

```

Abbildung 12 - XML Datenstruktur der Diagramme

4.2.1.3 Diagramm

```

<?xml version="1.0" encoding="UTF-8"?>
<diagramm name="DiagramName" type="FlowDesign" parent="root" scrollPositionX="500.0" scrollPositionY="300.0" activeVersionNum="1">
    <versions></versions>
</diagramm>

```

Abbildung 13 - Ausschnitt aus der XML Datenstruktur eines Diagramms

Im äußersten XML Tag werden Informationen über das Diagramm gespeichert. Dazu gehören zuerst einmal der Name des Diagramms und was für eine Diagrammart es ist. Mögliche Diagrammarten sind das System Umwelt Diagramm, das Dialog Diagramm und das Flow Design Diagramm.

In der Software kann man jedem Diagramm ein Elterndiagramm zuweisen. Das hat den Vorteil, dass die Diagramme anschließend hierarchisch in der Baumstruktur angezeigt werden können. Der Name des Elterndiagramms wird ebenfalls in dem Diagramm-Tag abgespeichert. Ist kein Elterndiagramm gesetzt, wird automatisch Root eingetragen. In der Baumstruktur sind diese Diagramme dann nach der Diagrammart sortiert dargestellt.

Da wir in der Lage sind mehrere Versionen eines Diagramms abzuspeichern, ist es auch wichtig, dass man hier abspeichert, in welcher Version man gerade arbeitet.

Wichtig: Die Scroll-Positionen sind in der Datenstruktur enthalten und werden auch in die Software eingelesen bzw. auch wieder abgespeichert, haben in der Software aber noch keine Anwendung. Die Überlegung hierzu war, dass man die letzte Position der Zeichenfläche abspeichert, um beim laden des Diagramms die Zeichenfläche automatisch ausrichten zu können.

Das Diagramm-Tag enthält als Kind-Elemente sämtliche Versionen des Diagramms.

4.2.1.3.1 Version

```
<version num="1" created="DDMMJJJJ" edited="DDMMJJJJ">
    <nodes></nodes>
    <wires></wires>
</version>
```

Abbildung 14 - Ausschnitt aus der XML Datenstruktur eines Diagramms

Jedes Diagramm hat mindestens eine Version, die bei der Erstellung automatisch angelegt wird.

Für jede Version wird die Versionsnummer abgespeichert, das Datum wann die Version angelegt wurde und auch wann die Version zuletzt bearbeitet wurde.

Jede Version enthält Nodes und Wires. Nodes sind zum Beispiel Funktionseinheiten oder Ressourcen. Wires sind die Pfeile, mit denen man zwei Nodes verbinden kann.

4.2.1.3.2 Node

```
<node id="1" type="nodeType" x="12.9" y="36.0" width="120.4" height="65.2">
    <text>Text</text>
    <label>Label</label>
    <attribut>S</attribut>
    <link>Link</link>
    <image>URL</image>
    <color>blue</color>
    <additional>Text</additional>
    <connectors></connectors>
</node>
```

Abbildung 15 - Ausschnitt aus der XML Datenstruktur eines Diagramms

Jede Node hat eine eindeutige ID, über die sie identifiziert werden kann. Die IDs sind 32-bit lange Zeichenketten, die mithilfe der UUID-Funktion in Java erzeugt werden.

Für jede Node wird auch der Type abgespeichert. Wir sind in der Lage sämtliche Varianten von Nodes in dieser Datenstruktur abzuspeichern. Die Diagrammart spielt dabei keine Rolle. Die Node Typen sind in der Software als Enums implementiert und somit eindeutig.

Jede Node kann in der Software auf der Zeichenfläche platziert werden. Die genaue Position wird sowie die Größe der Node wird ebenfalls in der Datenstruktur gespeichert.

Neben diesen allgemeinen Informationen, welche für jede einzelne Node abgespeichert werden, gibt es auch noch weitere Werte, die je nach Type der Node benötigt werden. Dazu gehören ein Text und ein Label, ein Attribut, welches auch als Enum in der Software vorhanden

ist, einen Link auf auf ein anderes Diagramm, ein Bild, eine Farbe und ein Text, welcher für seltene Spezialfälle verwendet wird.

Als letzte wichtige Information, werden für jede Node sogenannte Connectoren (Verbindungspunkte) abgespeichert. Diese Verbindungspunkte sind von elementarer Bedeutung für die Arbeitsweise der Software. Jede Node hat eine beliebige Anzahl solcher Verbindungs punkte an denen die Wires andocken können. Notwendig sind sie, um den Datenfluss innerhalb des Diagramms prüfen zu können.

4.2.1.3.3 Connector

```
<connector id="1" orientation="left" type="string" />
```

Abbildung 16 - Ausschnitt aus der XML Datenstruktur eines Diagramms

Jeder Connector (Verbindungspunkt) hat eine eindeutige ID, für die wir auch wieder UUIDs aus Java einsetzen.

Über die Orientierung wird abgespeichert, an welcher Seite ein solcher Verbindungspunkt an der Node sitzt (oben, unten, rechts oder links). Dadurch ist auch gleich festgelegt, ob es sich um ein Input oder ein Output Connector handelt. Von links kommen Daten herein und nach rechts fließen sie wieder heraus. Nach unten werden Ressourcen angehängt, welche bei den Ressourcen selber von oben angedockt werden.

Bei dem Typ handelt es sich um den Datentyp. Jeder Verbindungspunkt hat einen eindeutigen Datentyp, bzw. mehrere Datentypen. Darüber kann man beim Verbinden zweier Nodes überprüfen, ob der Output Datentyp der ersten Node gleich dem Input Datentyp der zweiten Node ist. Das ist notwendig, um einen korrekten Datenfluss zu gewährleisten.

4.2.1.3.4 Wire

```
<wire id="1" type="wireType" srcConn="connectionID" targetNode="nodeID" targetConn="connectionID">
    <label>Text</label>
    <joints></joints>
</wire>
```

Abbildung 17 - Ausschnitt aus der XML Datenstruktur eines Diagramms

Eine Wire beschreibt die Verbindung zwischen zwei Nodes (von Connector zu Connector). Dabei hat jede Verbindung eine eindeutige ID, welche auch wieder über die UUID Funktion von Java erzeugt wird.

Es gibt unterschiedliche Arten von Verbindungen. Zum Beispiel gibt es eine Verbindung zwischen Input und Output Connector zweier Nodes, über die Daten ausgetauscht werden. Eine andere Art von Verbindung ist das Verknüpfen einer Funktionseinheit mit einer Ressource.

Um diese verschiedenen Verbindungen voneinander unterscheiden zu können, hat jede Verbindung einen eindeutigen Typ, welcher als Enum in der Software hinterlegt ist.

Um genau zu wissen, von wo nach wo eine Verbindung ist, werden die entsprechenden Nodes und der verwendete Connector in der Verbindung gespeichert.

Eine Verbindung kann zusätzlich eine Beschriftung erhalten, welche auch in der Datenstruktur gespeichert wird.

Joints sind Gelenkpunkte der Wire. Mit diesen ist es möglich, nicht nur gerade Pfeile zeichnen zu können, sondern diese zwischen durch auch abknicken zu können.

Wichtig: Joints sind in der Datenstruktur eingebaut und werden sowohl in die Software geladen, als auch gespeichert. Momentan besteht in der Software aber noch keine Möglichkeit diese in der Zeichenfläche setzen zu können.

4.2.1.3.5 Joint

```
<joint x="12.3" y="32.1" />
```

Abbildung 18 - Ausschnitt aus der XML Datenstruktur eines Diagramms

Ein Joint ist nur ein sehr einfaches Element. Die einzigen Informationen die hier gespeichert werden sind die genauen Positionsdaten auf der Zeichenfläche.

4.2.2 Ordnerstruktur

Die Software erzeugt beim Anlegen eines neuen Projekts automatisch die notwendige Ordnerstruktur. Im Folgendem wird kurz darauf eingegangen, wie die Dateien auf der Festplatte angelegt werden.

Das Projektverzeichnis enthält die oben beschriebene Projektdatei und zwei Ordner.

Der erste Ordner heißt **Diagramme** enthält weitere drei Ordner, für jeden Diagrammtyp einen. In diesen drei Ordner liegen die XML-Dateien der Diagramme.

Der zweite Ordner heißt **Ressourcen**. In diesem Ordner befindet sich momentan nur ein weiterer Ordner, in dem sämtliche Bilder enthalten sind, welche in das Projekt importiert wurden. In der Zukunft ist es aber möglich, hier noch weitere Ressourcen abzuspeichern.

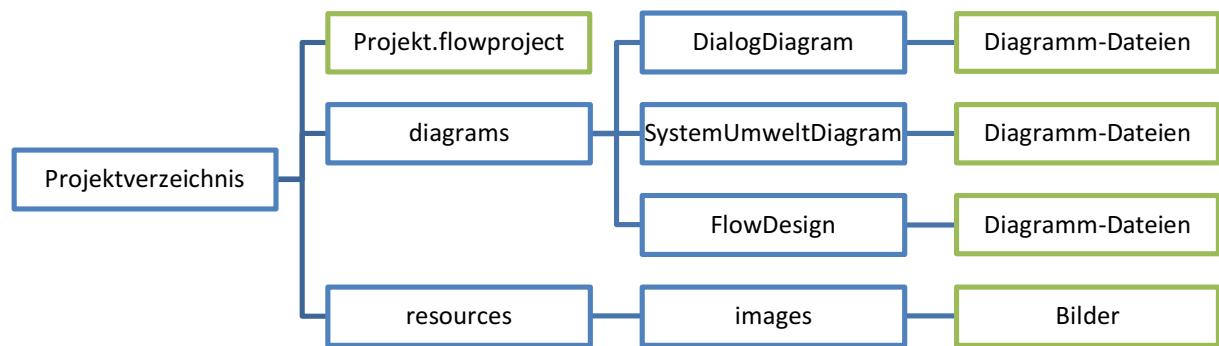


Abbildung 19 - Ordnerstruktur eines Projekts

4.2.3 Java Klassen

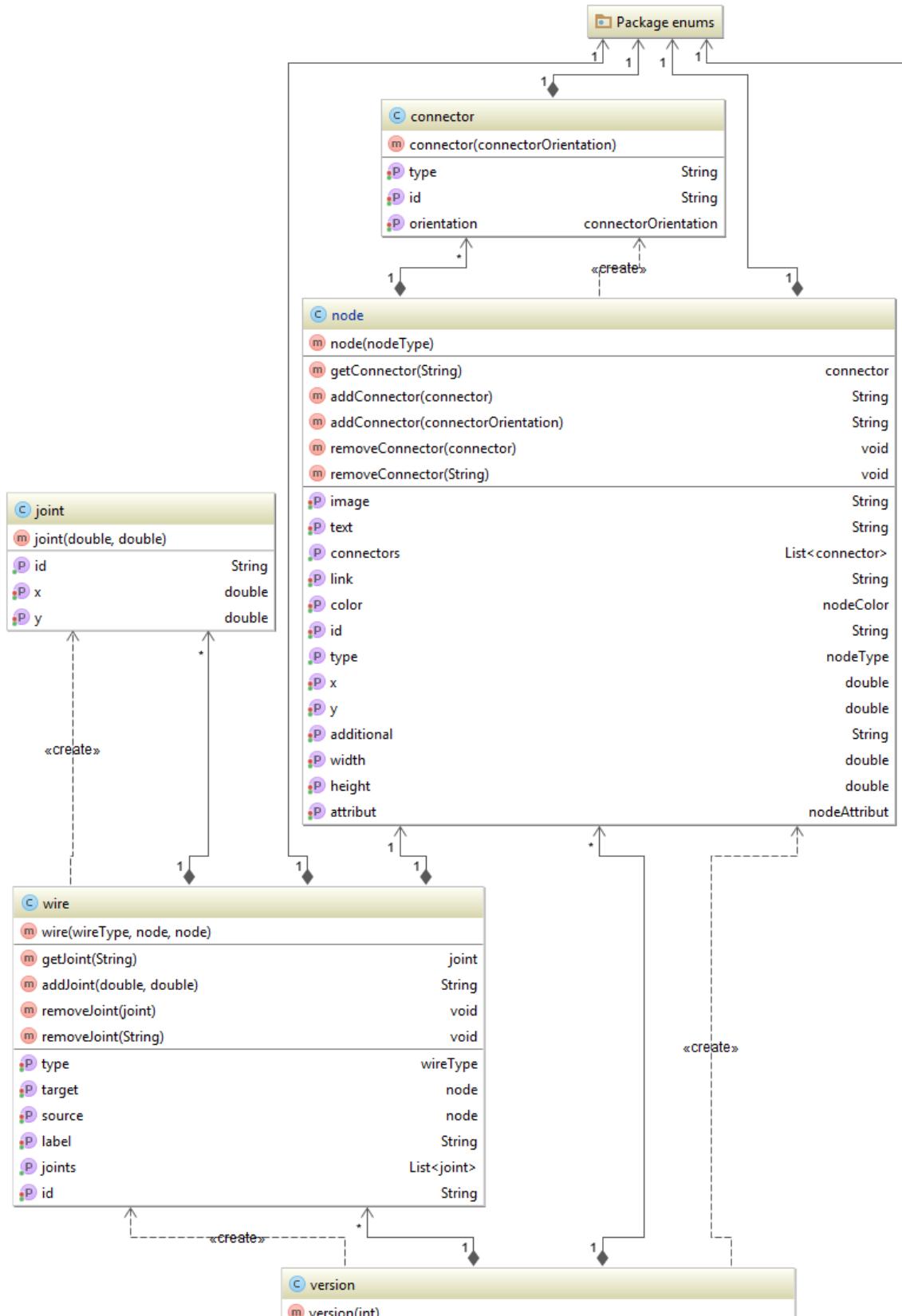


Abbildung 20 - Java Class Diagramm 1

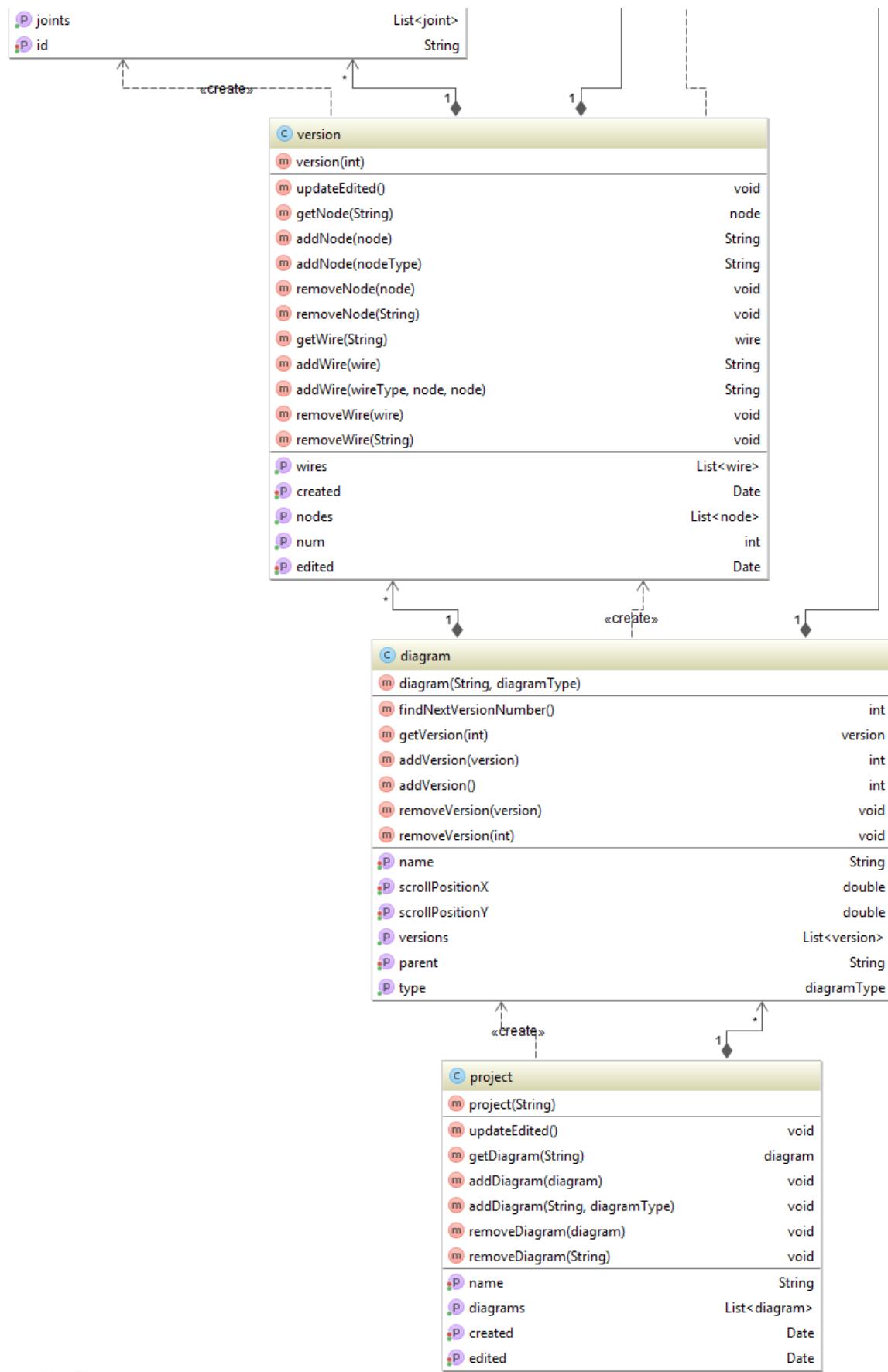


Abbildung 21 - Java Class Diagramm 2

Die Java Klassen sind an Hand der XML Datenstruktur umgesetzt. Auch hier in den Java Klassen wird noch nicht zwischen den unterschiedlichen Diagramm- und Node Arten unterschieden. Die Unterscheidung passiert erst im Frontend, wo es für jede Node eine eigene Klasse gibt, in der die unterschiedlichen Charakteristiken und ihr Aussehen definiert sind.

Durch die einheitliche Datenstruktur sind wir in der Lage auf einfachste Art und Weise und völlig unabhängig von der Art der Node, mit der Datenstruktur zu arbeiten. Die Zugriffe sehen dafür immer gleich aus und so können wir eine einheitliche API zur Verfügung stellen.

4.2.4 Enums

Im Modul Daten sind auch Enums definiert, welche über sämtliche Module hinweg einheitlich genutzt werden. Im Folgende werden die Enums kurz beschrieben.

4.2.4.1 Diagrammart

In diesem Enum sind die Namen der drei Diagrammarten System Umwelt Diagramm, System Umwelt Diagramm und Flow Design definiert.

4.2.4.2 Nodes

Alle Nodes, welche in der Anwendung zur Verfügbar stehen, sind hier aufgelistet. Die Namen sind mit einem Präfix versehen, über den sie zu einer Diagrammart zugeordnet werden können.

4.2.4.3 Farbe der Node

Einigen Nodes kann man eine Hintergrundfarbe geben. Alle zur Verfügung stehenden Farben sind hier definiert.

Wichtig: Momentan speichern wir den Namen der Farbe in der Datenstruktur ab (zum Beispiel die Farbe Rot). Im Frontend wird der Name dann in RGB-Farben übersetzt. In der Zukunft wäre denkbar diesen Schritt zu überspringen und direkt den RGB bzw. einen HEX-Wert abzuspeichern.

4.2.4.4 Attribut der Node

Funktionseinheiten können Attribute wie zum Beispiel Singelton oder Multiton zugewiesen bekommen. Alle verfügbaren Attribute sind in diesem Enum definiert.

4.2.4.5 Connector Orientation

Connectoren können an unterschiedlichen Seiten einer Node platziert werden. Je nach Seite haben die Connectoren unterschiedliche Bedeutungen, zum Beispiel ist links ein Input und rechts ein Output.

4.2.4.6 Wires

In der Anwendung kommen verschiedene Verbindungsarten vor. Zum Beispiel gibt es einen Datenstrom und eine Abhängigkeit von einer Ressource. Was für verschiedene Wires es gibt, ist hier definiert.

4.2.5 Testing

Die gesamte Software arbeitet auf Basis dieser Datenstruktur. Daher war es uns wichtig, dass die Java Klassen der Datenstruktur korrekt implementiert sind. Daher haben wir für das Modul Daten J-Unit Tests geschrieben.

Da wir im Laufe der Entwicklung noch das eine oder Feld in der Datenstruktur ergänzen mussten, haben sich die J-Unit Tests sehr ausbezahlt. So waren wir in der Lage die Datenstruktur nach einer Änderung auf Knopfdruck zu testen und eventuell auftretende Fehler sofort zu erkennen und zu beheben.

4.3 Backend

4.3.1 Datenspeicher

Im Modul Backend befindet sich der Datenspeicher „**DataStorage**“, in dem alle wichtigen Daten hinterlegt sind.

4.3.1.1 Projektpfad

Im Projektpfad ist der Pfad des Projektverzeichnisses auf der Festplatte gespeichert.

4.3.1.2 Projekt

Wird ein Projekt von der Festplatte geladen, dann wird das Projekt im Datenspeicher abgelegt. Ab diesem Moment wird nur noch beim Speichern auf die Festplatte geschrieben.

4.3.1.3 Aktives Diagramm

Im Datenspeicher liegt auch das zurzeit aktive Diagramm, bzw. eine Referenz auf das entsprechende Diagramm im Projekt. Das aktive Diagramm ist das, mit dem man aktuell auf der Zeichenfläche interagiert. Die meisten der Schnittstellen im Backend kommunizieren direkt mit dem aktiven Diagramm.

4.3.1.4 Speicherpfad

Der Speicherpfad ist der Pfad einer XML-Datei, welche im Softwareverzeichnis liegt (backend/resources/). In dieser Datei werden Dinge abgespeichert, welche auch nach Programmneustart noch vorhanden sein sollen. Dazu gehören zum Beispiel die zuletzt geöffneten Projekte oder auch vom Benutzer definierte Datentypen.

4.3.1.5 Zuletzt geöffnete Projekte

Im Datenspeicher sind die zuletzt geöffneten Projekte hinterlegt, um diese im Startdialog-Fenster anzeigen zu können.

4.3.1.6 Canvas Spaces

Die Canvas Spaces ist eine Java-Map, welche die geöffneten Diagramme mit der zugeordneten Zeichenfläche verknüpft. Wird ein Diagramm geöffnet, wird automatisch eine neue Zeichenfläche (Canvas Space) generiert und dem Diagramm zugewiesen. Auf der Zeichenfläche wird dann das Diagramm ausgegeben und kann dort bearbeitet werden. Wird ein Diagramm geschlossen, wird auch der entsprechende Eintrag in der Java-Map entfernt.

4.3.1.7 Datentypen

Im Datenspeicher werden auch die Datentypen hinterlegt, welche im Frontend zur Verfügung stehen. Dabei sind die Primitiven Datentypen standartmäßig verfügbar, der Nutzer kann aber auch selber neue Datentypen hinzufügen und sie mit einer Beschreibung versehen. Diese Datentypen dienen auch dazu, den Datenfluss zwischen zwei Nodes zu validieren.

4.3.1.8 Änderungsliste

Wird ein Diagramm bearbeitet, wird es in die Änderungsliste eingetragen. Beim nächsten Mal, wenn das Projekt gespeichert wird, werden alle Diagramme, die auf der Änderungsliste stehen gespeichert. Dadurch wird sichergestellt, dass nicht jedes Mal alle Diagramm abgespeichert werden müssen.

4.3.2 XML Parser

Das Backend stellt zwei XML Parser zur Verfügung. Der erste schreibt die Daten als XML auf die Festplatte und der andere liest sie von dort wieder ein.

4.3.2.1 XML schreiben

Der XML Parser der die Daten als XML abspeichert, speichert aktualisiert zuerst die Projektdatei im Projektverzeichnis.

Anschließend geht der Parser jedes Diagramm im Projekt durch und überprüft an Hand der Änderungsliste im Datenspeicher, ob das Diagramm geändert wurde und speichert es gegebenenfalls ab.

Zudem enthält der Parser auch eine Funktion, um den Datenspeicher in die softwareinterne XML-Datei zu schreiben. Diese Funktion wird beim Speichern des Projekts aber nicht automatisch aufgerufen.

4.3.2.2 XML lesen

Der XML Parser zum Lesen von XML Dateien ist so aufgebaut, dass er ein ganzes Projekt einlesen kann, oder auch nur eine einzelne Diagrammdatei. Dadurch ist es möglich den Parser auch zu verwenden ein einzelnes Diagramm in das Projekt zu importieren.

Der Parser enthält eine Funktion, um die programminterne XML Datei in den Datenspeicher einzulesen. Auch diese Funktion muss manuell aufgerufen werden, da dies nur einmal beim Programmstart notwendig ist, währenddessen man zur Laufzeit beliebig oft Projekte laden und speichern kann.

4.3.3 Schnittstellen

Um auf die Daten im Datenspeicher zugreifen zu können, gibt es Funktionen, welche alle über die Backend-API zugänglich sind.

Die Backend-API enthält selber nur einige kleine Funktionen, verweist aber auf alle anderen verfügbaren APIs. Im Folgenden ist eine Auflistung an verfügbaren APIs.

4.3.3.1 Backend-API

Enthält alle anderen APIs. Außerdem gibt es Funktionen um den Datenspeicher zu initialisieren, um auf die Datentypen zuzugreifen und um Diagramm in die Änderungsliste einzutragen.

4.3.3.2 File-API

Die File-API stellt Funktionen bereit, um mit Dateien von der Festplatte zu arbeiten. Im Wesentlichen beschränken sich die Funktionen dabei auf den Umgang mit Bildern, welche in das Projekt importiert werden können.

4.3.3.3 Project-API

Über die Projekt-API kann man Projekte erstellen, speichern und öffnen. Außerdem biete sie Funktionen, um aus den im Projekt enthaltenen Diagrammen eine hierarchische Baustuktur zu erzeugen.

4.3.3.4 Diagramm-API

Über die Diagramm-API kann man mit einem einzelnen Diagramm interagieren. Man kann verschiedene Versionen laden, neue Versionen erstellen oder löschen, man kann das aktive Diagramm setzen, Diagramme erstellen oder löschen etc.

4.3.3.5 Node-API

Die Node-API stellt verschiedene Funktionen zur Verfügung, um mit einer einzelnen Node zu arbeiten. Man kann neue Nodes erstellen, kann bestehende entfernen, oder kann die verschiedenen Werte einer Node verändern.

4.3.3.6 Connector-API

Über die Connector-API interagiert man mit einem einzelnen Connector. Es stehen Funktionen zur Verfügung, um Connectoren hinzuzufügen, sie zu entfernen oder den Datentyp zu verändern.

4.3.3.7 Wire-API

Die Wire-API stellt Funktionen bereit, um auf eine einzelne Wire zuzugreifen. Neben dem Erstellen und Löschen von Nodes, kann man auch das Start und Ziel der Verbindung festlegen.

4.3.3.8 Joint-API

Über die Joint-API greift man auf einen einzelnen Joint zu. Man kann einen Joint erstellen, löschen, oder seine Position verändern.

4.4 Frontend

Bei der Umsetzung des Frontend-Oberfläche, wurde auf das Konzept einer Border-Pane aufgebaut. Eine Border-Pane beschreibt Regionen, die einen bestimmten Bereich der Maske darstellen. So wird die Erstellung einer übersichtlichen Oberfläche vereinfacht.

4.4.1 Zeichenfläche

4.4.1.1 Grundlegender Aufbau der Zeichenfläche

Die Position der Zeichenfläche liegt in der Center-Region der Border-Pane. Die Zeichenfläche selbst ist als eine Pane definiert. Um das Scrollen und Zoomen zu ermöglichen wurde diese zusätzlich in einer Scroll-Pane verpackt. Die Zeichenfläche hat zum Zeitpunkt des Erstellens keine Größe. Die Größe wird erst durch Zeichenflächenelemente, welche dort abgelegt werden können, bestimmt.

4.4.1.2 Zeichenflächenelemente

Auf der Zeichenfläche können Elemente platziert werden. Im Allgemeinen lassen sich die Elemente in folgende zwei Typen einteilen:

4.4.1.2.1 Nodes

Als Nodes werden die Elemente bezeichnet, die auf die Zeichenfläche gezogen werden können. Welche Arten von Nodes es gibt, kann im „CheatSheet-Flow-Design“⁶ nachgesehen werden. Sie stellen die Basis zum Erstellen eines Diagramms dar. Jede Node kann mit Verbindungspunkten ausgestattet werden, welche die Start- und die Endpunkte von Wires darstellen.

⁶ [online] <http://refactoring-legacy-code.net/wp-content/uploads/2016/10/CheatSheet-Flow-Design.pdf>

4.4.1.2.2 Verbindungspunkte

In unserer Anwendung können Verbindungen nur über Verbindungspunkte aufgebaut werden. Je nach Typ und Ausrichtung des Verbindungspunktes wird ein Datenfluss oder eine Abhängigkeit dargestellt. Im folgendem eine kurze Zusammenfassung welche Verbindungspunktformen im welchem Diagramm verwendet werden und welchen Zweck sie dabei erfüllen. Die angegebenen Bezeichnungen entsprechen den im Code umgesetzten Verbindungspunkten (Connector).

Bezeichnung	Form	Position	Zweck	Verwendet in
Point Connector		Auf jeder Seite einer Node	Stellt den Input und Output einer Abhängigkeit dar	System Umwelt Diagramm
Top Connector		Über einer Node	Stellt den Input einer Abhängigkeit dar	Flow Design Diagramm
Bottom Connector		Unter einer Node	Stellt den Output einer Abhängigkeit dar	Flow Design Diagramm
Left Connector		Links von einer Node	Stellt den Input eines Datenflusses dar	Flow Design Diagramm
Right Connector		Rechts von einer Node	Stellt der Output eines Datenflusses dar	Flow Design Diagramm

Tabelle 1 - Verbindungspunkte

4.4.1.2.3 Wires

Als Wires werden die Linien bezeichnet, welche die einzelnen Nodes miteinander verbinden. Die verschiedenen Arten von Wires können ebenfalls in der beigefügten Datei „CheatSheet-Flow-Design“ eingesehen werden.

4.4.1.3 Umsetzung im Code

Die Umsetzung einer Node, einer Wires oder eines Verbindungspunktes wurde über abstrakten Klassen realisiert. Im Folgenden sind die verwendeten abstrakten Klassen, die für die Zeichenfläche verwendet wurden aufgelistet.

4.4.1.3.1 Abstrakte Klasse für Nodes

Die abstrakte Klasse für Nodes ist ebenfalls, wie die Zeichenfläche auch, als eine Pane definiert. So ist es möglich definierte Formen in einer Node zu erstellen und zu positionieren.

Für die abstrakte Klasse der Nodes (im Code „Element“) sind unteranderem folgende Eigenschaften definiert:

- Eine ID zur eindeutigen Identifikation
- Typ einer Node
- Position (horizontaler und vertikale Position)
- Größe (Höhe und Breite)
- Farbe (mögliche Farben sind im einem ENUM im Backend definiert)
- Text
- Label
- Attribute (mögliche Attribute sind im einem ENUM im Backend definiert)
- Link
- Image
- Liste aller Verbindungspunkten einer Node
- Anzahl für jeden einzelnen Verbindungspunkttyp
- Rahmen (Jede Node hat einen Rahmen, speziell auf die Nodegröße angepasst)

4.4.1.3.2 Abstrakte Klasse für Wires

Eine Wire stellt eine Gruppe dar, die je nach Form des Wires eine Verbindung aus einem Dreieck und einer Linie (Datenfluss) oder aus einem Punkt und einer Linie (Abhängigkeit) besteht.

Die abstrakte Klasse der Wires (im Code ebenfalls „Wires“) enthält folgende Eigenschaften:

- Eine ID zur eindeutigen Identifikation
- Ziel- und Start-Verbindungspunkt
- Ziel- und Start-Node
- Typ des Wire

4.4.1.3.3 Abstrakte Klasse für Verbindungspunkte

Die Verbindungspunkte, welche auf einer Node platziert werden können, sind ebenfalls als eine Pane definiert. Je nach Verbindungspunkttyp haben Verbindungspunkte eine andere Form (Kreis oder Dreieck). Genauere Informationen über Verbindungspunkte sind in der Tabelle „Verbindungspunkte“ vorzufinden.

Die abstrakte Klasse der Verbindungspunkte (im Code „Connector“) haben folgende Eigenschaften:

- Eine ID zur eindeutigen Identifikation
- Die zugehörige Node
- Die Ausrichtung des Verbindungspunktes
- Eigenschaften für den Datentyp

Da jede Node, jede Wire und jeder Verbindungspunkt andere Eigenschaften haben, wurde für jeden Typ eine eigene Klasse angelegt. Diese legen die Form eines Elements fest und definieren dessen Standardeigenschaften.

4.4.2 FileTree

4.4.2.1 Grundlegender Aufbau

Der FileTree liegt in der „Left-Region“ der Border-Pane. Der FileTree selbst benutzt die „TreeView“ Klasse, welche von JavaFX zur Verfügung gestellt wird. Diese Klasse wird in JavaFX dafür verwendet ein Dateiverzeichnis grafisch darzustellen. Weiterhin beinhaltet der TreeView nun unterschiedliche „TreeItems“.

Das Root-TreeItem ist dabei ein Ordner der den gleichen Namen wie das Projekt hat. Darunter liegen die Ordner der einzelnen Diagrammarten. In diesen liegen wiederum die TreeItems mit den Diagrammnamen.

Eine Besonderheit gibt es bei den Flow Design Diagrammen. Hier ist es möglich, dass ein Diagramm wiederum ein Kind-Diagramm hat. Dieses Kind-Diagramm wird dann unter seinem Eltern-Diagramm angezeigt. Dies gilt allerdings nur für den Verzeichnisbaum auf der Software Seite. Auf der Festplatte sind auch die Flow-Diagramme nur in einem Ordner abgelegt und beinhalten in ihrer XML-Struktur einen Verweis auf das Eltern-Diagramm.

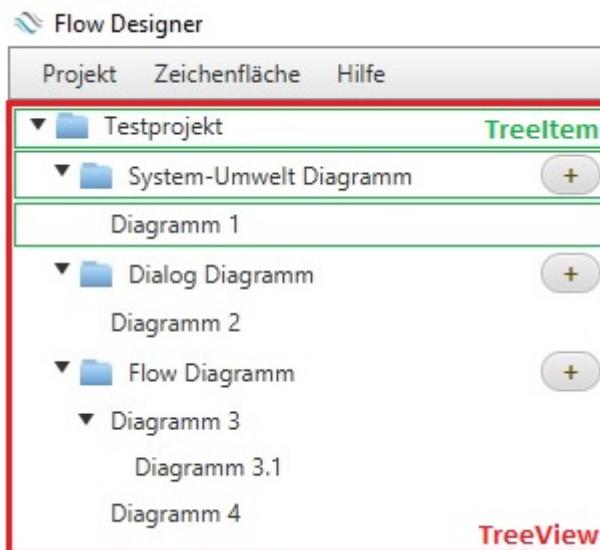


Abbildung 22 – Treeview

Die TreeItems werden mit einem Objekt der Klasse „CustomItem“ erstellt, da das TreeItem selber keine großartige Möglichkeit bietet, seinen Inhalt zu bestimmen. Das „CustomItem“ erweitert hierbei eine HBox und je nach Verwendungszweck, wird dieser HBox z.B. ein Ordner Icon, ein Text, der "+" Button oder ein verstecktes Attribut, welches den Diagrammtyp bestimmt, hinzugefügt.

4.4.2.2 Funktion

- Mit einem Doppelklick auf ein Diagramm öffnet sich dieses in der Zeichenfläche. Hierzu wurde in der „init()“-Funktion des FileTrees ein „Handler“ auf einen Doppelklick gesetzt.
- Um den FileTree zu aktualisieren kann die Funktion „updateFileTree()“ aufgerufen werden. Diese holt sich dann automatisch vom Backend, den Verzeichnisbaum des aktuell geöffneten Projekts.
- Mit einem Rechtsklick auf ein TreeItem öffnet sich ein dem TreeItem entsprechendes Kontextmenü.

4.4.2.3 Kontextmenü

Das Kontextmenü wird auf den TreeView angewendet. Wir verwenden auch hier die von JavaFX zur Verfügung gestellte Klasse „ContextMenu“. Je nachdem ob nun ein Projekt geöffnet ist oder nicht, zeigt das Kontextmenü unterschiedliche Optionen an.

Außerdem gibt es noch unterschiedliche Optionen, je nachdem welches TreeItem angeklickt wurde. Hierzu wird bei einem Rechtsklick ein Event-Handler ausgelöst, der das Kontextmenü anhand von dem angeklickten TreeItem updatet und dann das Kontextmenü auf dem Bildschirm anzeigt.

Die Funktion das gesamte Projekt zu löschen ist momentan immer ausgegraut, da dies noch nicht implementiert ist und momentan manuell über einen externen Datei-Explorer gemacht werden muss.

4.4.3 Statusbar

Die Statusbar ist dafür gedacht, dass das Programm dem Benutzer Zustände oder Informationen mitteilen kann. Dies kann zum Beispiel sein, wenn ein Projekt gespeichert wurde oder aber auch wenn ein Fehler auftritt.

4.4.3.1 Grundlegender

Die Statusbar liegt in der „Bottom-Region“ der „Border-Pane“. Die Statusbar selber ist eine „HBox“ welche von JavaFX zur Verfügung gestellt wird. In dieser „HBox“ befindet sich nun lediglich eine JavaFX „Text“-Komponente, welche mittig und nach links ausgerichtet wird.

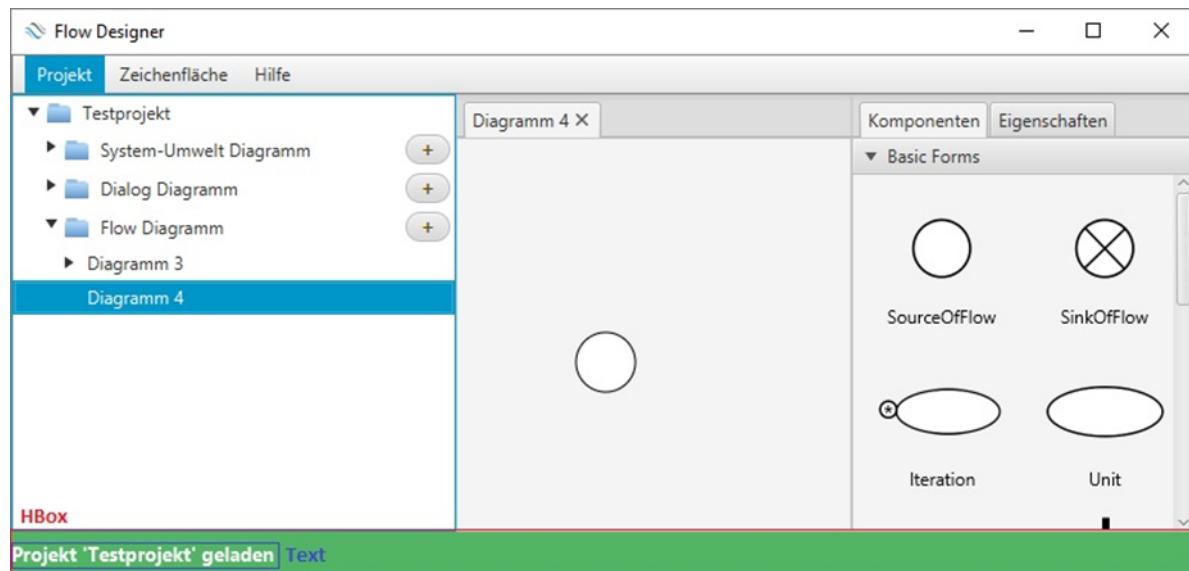


Abbildung 23 – Statusbar

4.4.3.2 Funktion

Die Statusbar hat 3 verschiedene Funktionen:

- Sie kann Erfolgs-Meldungen (grün) anzeigen: **displaySuccess(String message)**.
- Sie kann Warnung-Meldungen (orange) anzeigen: **displayWarning(String message)**.
- Und sie kann eine Error-Meldung (rot) anzeigen: **displayError(String message)**.

Jede Meldung wird für vier Sekunden angezeigt und verschwindet dann automatisch. Diese Zeit kann in der jeweiligen Methode angepasst werden. Außerdem kann die Statusbar nach Belieben erweitert werden, indem weitere Funktionen eingefügt werden.

4.4.4 Menüleiste

4.4.4.1 Grundlegender

Die Menüleiste liegt in der „Top-Region“ der „Border-Pane“. Die Menüleiste verwendet die von JavaFX zu Verfügung gestellte Klasse „MenuBar“. Diese „MenuBar“ beinhaltet wiederum mehrere Instanzen der Klasse „Menu“, welche ebenfalls von JavaFX zu Verfügung gestellt wird. Ein „Menu“ beinhaltet letztendlich „MenuItems“, welche bei einem Linksklick Operationen ausführen.

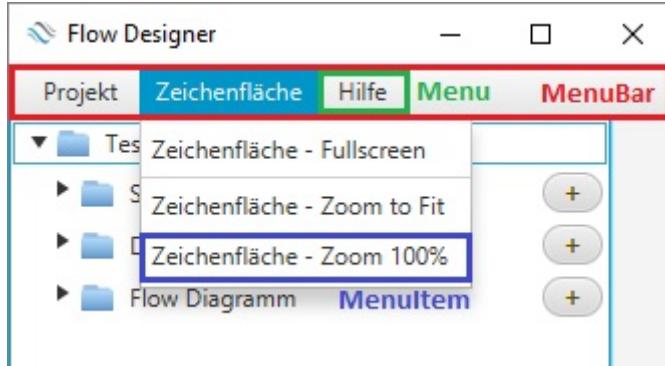


Abbildung 24 - Menüleiste

4.4.4.2 Funktion

Die Menüleiste hat drei verschieden Menüpunkte welche nun im Einzelnen erläutert werden.

- Menüpunkt „Projekt“
 - o Ein Projekt kann erstellt werden (ruft **createProject()** auf)
 - o Ein Projekt kann geöffnet werden (ruft **openProject()** auf)
 - o Ein Projekt kann gespeichert werden (ruft **saveProject()** auf)
 - o Ein Projekt kann geschlossen werden (ruft **closeProject()** auf)
- Menüpunkt „Zeichenfläche“
 - o Die Zeichenfläche kann in den Fullscreen-Modus gesetzt werden (ESC um Fullscreen zu verlassen)
 - o Die Zeichenfläche kann so gezoomt werden, dass alle Komponenten auf einen Blick zu sehen sind
 - o Die Zeichenfläche kann auf 100% Zoom gesetzt werden (Standartwert)
- Menüpunkt „Hilfe“
 - o Beinhaltet ein Dialog Fenster, in dem alle verfügbaren Tastenkombinationen aufgelistet sind.

4.4.5 Komponenten und Eigenschaften

Die Komponenten und Eigenschaften unterteilen sich in zwei verschiedene Tabs und werden in zwei verschiedenen Java Klassen namens „FlowProperties“ und „Components“ beschrieben.

ben. Diese beiden Klassen werden durch die Klasse „RightSideBar“, welche sich in der „Right-Region“ der „Border-Pane“ befindet, als Tabs hinzugefügt.

4.4.5.1 Komponenten

4.4.5.1.1 Grundlegender Aufbau

Der Aufbau der Komponenten besteht aus einem „Accordion“.

Der Reiter Basic Forms beinhaltet ein „Grid“ zur Ausrichtung seiner Elemente. Die Elemente selber werden dabei als „Stackpane“ realisiert, die ein Bild des Elementes und dessen Namen beinhalten.

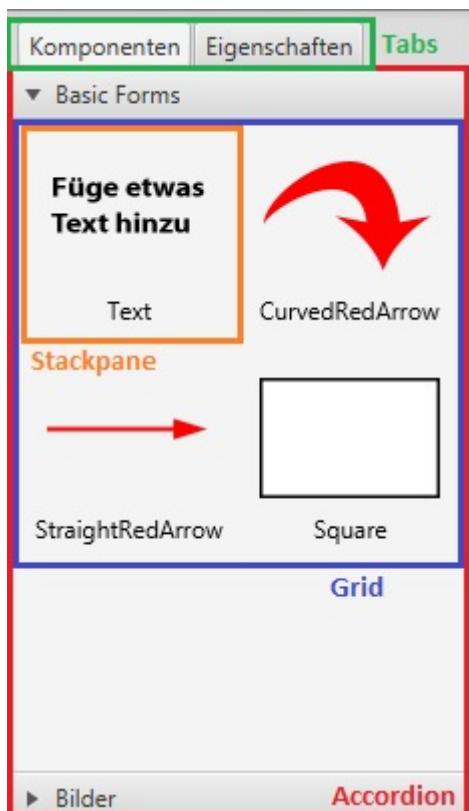


Abbildung 25 - Komponenten: Basic Forms

Der Reiter Bilder beinhaltet ebenfalls ein „Grid“ zur Ausrichtung der Elemente, darin wiederum einen „Button“ zum Import von Bildern und außerdem eine Liste von „Stackpanes“, mit bereits importierten Bildern.

4.4.5.1.2 Funktionen

Bei den Basic Forms kann per Drag&Drop ein Element in die Zeichenfläche gezogen werden. Dies wird durch einen „Event-Handler“, der auf dem „Stackpane“ vorhanden ist, realisiert. Beim Ziehen wird dabei ein „Dragboard“ erstellt. Dieses „Dragboard“ wird verwendet um zum einen das Drag&Drop-Event grafisch darzustellen und zum anderen um Informationen

des Elementes an die Zeichenfläche zu übergeben. Die Zeichenfläche fängt dieses „Dragboard“ dann auf und verarbeitet dieses.

Die selbe Funktionalität besteht ebenso bei den Bildern, so dass Bilder auf die Zeichenfläche gezogen werden können. Außerdem gibt es hier noch die Funktionalität ein Bild zu importieren. Wird ein Bild importiert so wird in der Klasse „Actions“ die Funktion „importImageToResource()“ aufgerufen.

4.4.5.2 Eigenschaften eines Diagramms

Wenn kein Element in der Zeichenfläche ausgewählt ist, so werden die Eigenschaften des Diagramms angezeigt. Diese bauen sich auf durch eine „VBox“, welche wiederum „VBoxen“ enthält welche verschiedene Einstellungsmöglichkeiten bieten.

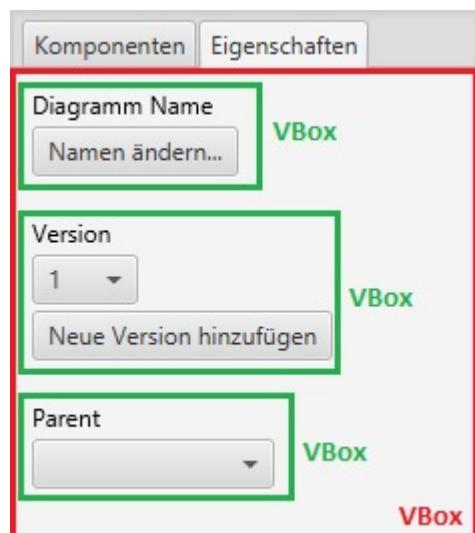


Abbildung 26 – Eigenschaften eines Diagrammes

4.4.5.2.1 Funktionen

- Der Name des Diagramms kann geändert werden.
- Es kann zwischen verschiedenen Diagrammversionen gewechselt werden.
- Es kann eine neue Version des Diagramms angelegt werden.
- Bei Flow-Diagrammen kann ein Parent angegeben werden, dem das Diagramm untergeordnet ist.

4.4.5.3 Eigenschaften einer Node

Ist ein Element in der Zeichenfläche ausgewählt, so werden die Eigenschaften des Elements angezeigt. Der Aufbau bleibt gleich wie bei den Diagrammeigenschaften und besteht aus einer Verschachtelung von „VBoxen“. Je nach Element werden außerdem andere Eigenschaften angezeigt, welche im Folgenden näher erläutert werden.

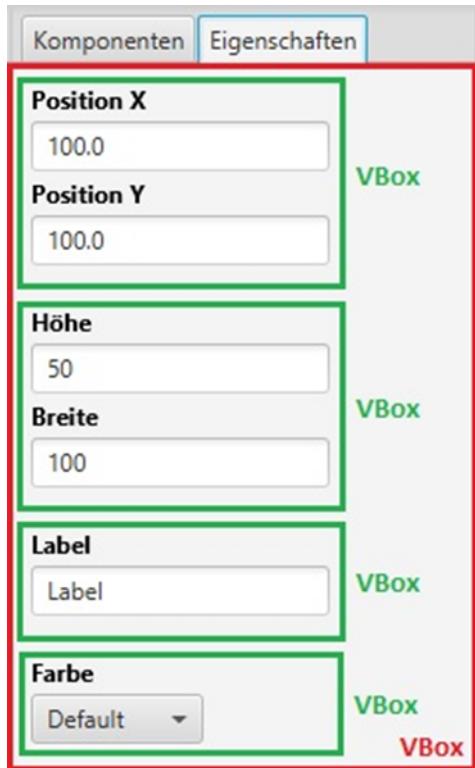


Abbildung 27 - Eigenschaften einer Node

4.4.5.3.1 Funktionen

- Die Position eines Elementes auf der Zeichenfläche kann verändert werden.
- Die Größe eines Elementes kann verändert werden.
- Das Label eines Elementes kann verändert werden.
- Die Farbe eines Elementes kann angepasst werden.
- Die Anzahl der Input- bzw. Output-Konnektoren kann erhöht oder verringert werden.
- Der Datentyp der Input- oder Output-Konnektoren kann verändert werden.
- Der Winkel mit dem ein Element angezeigt wird kann verändert werden.
- Der Text eines Elementes kann angepasst werden.
- Einem Element kann ein Attribut zugewiesen werden wie zum Beispiel Singleton.
- Ein Link auf ein anderes Diagramm kann gesetzt werden.

4.5 Projekt bauen

Die Software wird aus IntelliJ heraus gestartet. Die entsprechende Main-Funktion befindet sich im Modul Frontend in der Datei **App**.

4.5.1 Maven

Maven ist ein Tool um Abhängigkeiten aufzulösen. Alle Bibliotheken die man extern bezieht werden im Frontend in die sogenannte **POM** Datei eingetragen. Maven lädt diese dann automatisch herunter.

Wichtig: Da wir die Anwendung aus dem Frontend heraus starten ist es notwendig, auch Abhängigkeiten die in einem der anderen Module auftreten, in die POM Datei im Frontend einzutragen. Ansonsten werden diese Bibliotheken nicht in den Java ClassPath geschrieben und können nicht gefunden werden.

4.5.2 Language Level und Java Compiler

Die Anwendung setzt mindestens den **Java Compiler Version 1.8** voraus und benötigt Elemente aus dem **Java Language Level 8**.

Beide Werte müssen jedes Mal, nachdem Maven die Abhängigkeiten aktualisiert hat, gesetzt werden, da die Einstellungen hierfür durch den Maven Vorgang verworfen werden.

Die Einstellungen des Java Compilers sind unter den IntelliJ Einstellungen zu finden. Das Language Level wiederum muss in den Projekt Einstellungen angepasst werden.

5 Benutzerhandbuch

Im Folgenden werden die grundlegenden Elemente der Oberfläche erläutert. Um eine detaillierte Anleitung der verschiedenen Funktionen zu erhalten, schauen Sie bitte in das separat erhältliche Benutzerhandbuch.

5.1.1 Oberfläche

Unsere Software kann in fünf Abschnitte unterteilt werden:

1. Start Dialog Fenster

Das Start Dialog Fenster ist das erste Fenster, welches sich nach dem Starten des Programms öffnet.

2. Verzeichnisbaum

Der Verzeichnisbaum (FileTree) befindet sich auf der linken Seite im Hauptfenster der Software.

3. Zeichenfläche

Auf der Zeichenfläche werden alle Elemente eines Diagramms dargestellt und dem Benutzer ist es möglich hier mit diesen zu interagieren.

4. Eigenschaften und Komponenten

Diese beiden Bereiche befinden sich auf der rechten Seite des Hauptfensters unseres Programms. Zum einen findet man hier die Eigenschaften eines Diagramms oder einer ausgewählten Node. Zum anderen befindet sich hier der Bereich Komponenten, welcher alle Nodes enthält, die der Benutzer der Zeichenfläche einfügen kann.

5. Menüleiste

Ganz oben zieht sich durch das gesamte Hauptfenster die Menüleiste, in welcher man Einstellungen bezüglich des Projekts und der Zeichenfläche vornehmen kann oder Hilfe bezüglich der Software erhält.

5.1.1.1 Start Dialog Fenster

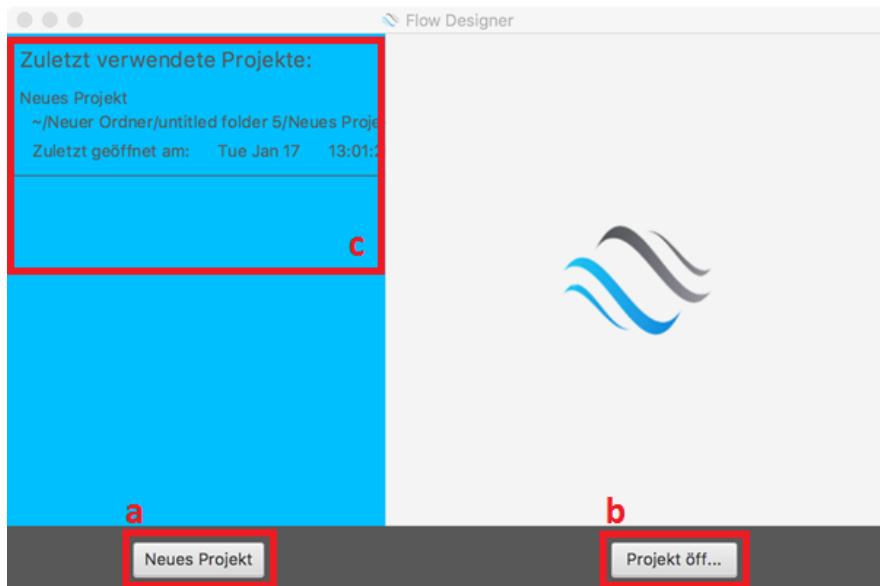


Abbildung 28 - Oberfläche: Start Dialog Fenster

Im Start Dialog Fenster hat der Anwender die Möglichkeit, ein neues Projekt anzulegen. Dafür muss ein Name und ein Speicherort ausgewählt werden (**a**).

Des Weiteren kann ein bestehendes Projekt importiert (**b**) oder eines der zuletzt verwendeten Projekte geöffnet werden (**c**).

5.1.1.2 Verzeichnisbaum

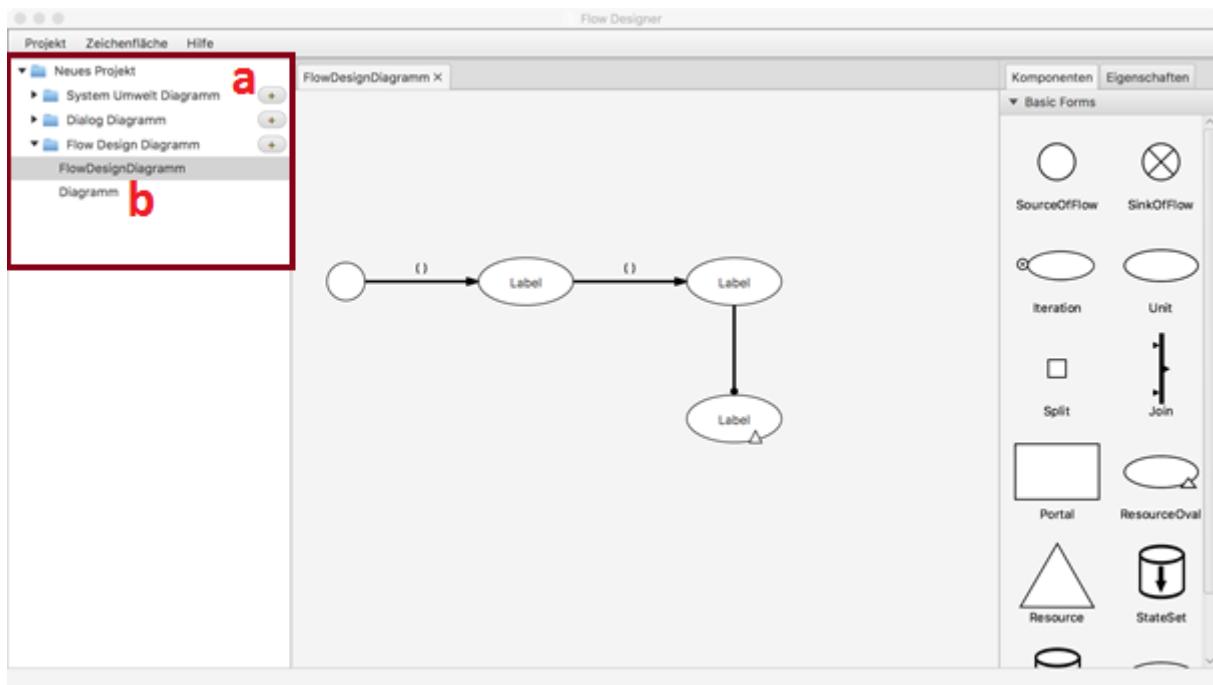


Abbildung 29 - Oberfläche: Filetree

Hier kann der Anwender ein neues Diagramm jeglicher Art anlegen (**a**). Dazu muss entweder der „+“-Button angeklickt, oder mittels eines Rechtsklicks auf die Diagrammart, die Option „Neues Diagramm“ ausgewählt werden. Ebenfalls hat er die Möglichkeit ein existierendes Diagramm zu öffnen, umzubenennen oder zu löschen (**b**). Zum Öffnen genügt ein Doppelklick auf den Namen. Das Umbenennen und Löschen geschieht mittels eines Rechtsklicks.

5.1.1.3 Zeichenfläche

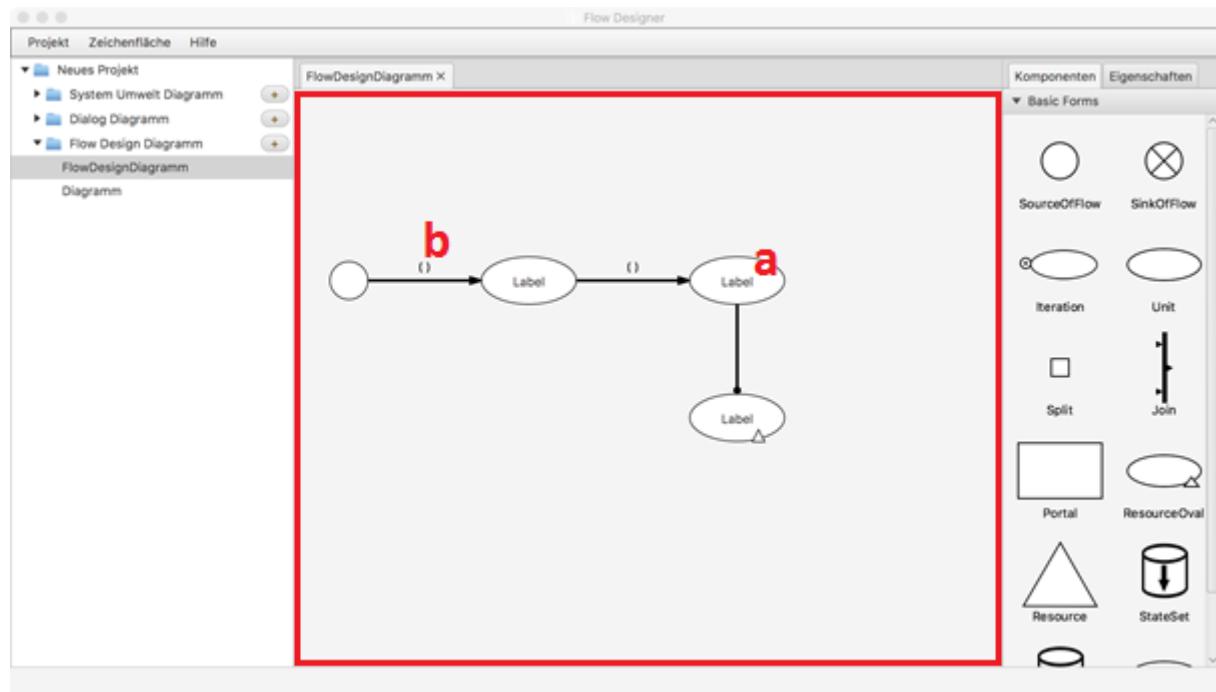


Abbildung 30 - Oberfläche: Zeichenfläche

Auf der Zeichenfläche steht die Interaktion des Benutzers mit den Elementen im Vordergrund. Aus diesem Grund lassen sich die wichtigsten Änderungen der Nodes direkt hier ausführen.

Zum Beispiel lassen sich Nodes (**a**) ganz einfach per Drag&Drop verschieben. Per Klick auf einen Datentyp (**b**) öffnet sich ein Fenster, in welchem man diesen bearbeiten kann. Auch kann eine Node (**a**) einfach per Rechtsklick und Auswahl der „Löschen“ Option gelöscht werden. Ebenfalls per Rechtsklick lässt sich ein neues Diagramm aus einer Node (**a**) generieren oder eine bestehender Link öffnen.

5.1.1.4 Eigenschaften und Komponenten

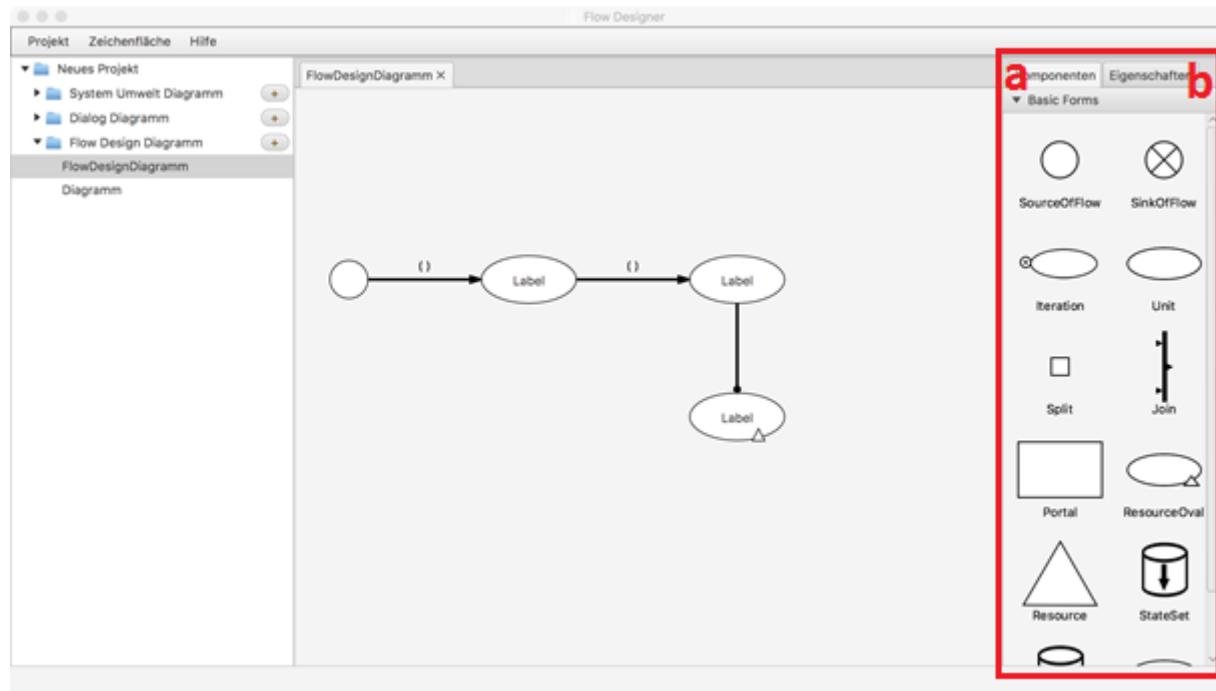


Abbildung 31 - Oberfläche: Eigenschaften und Komponenten

Da sich unser Programm an bestehenden Modellierungsprogrammen wie zum Beispiel Microsoft Visio orientiert, lassen sich aus dem Bereich Komponenten, Elemente der Zeichenfläche per einfaches Drag&Drop hinzufügen. Je nach geöffnetem Diagramm, werden hier die zulässigen Elemente angezeigt (**a**).

Sollte keine Node auf der Zeichenfläche ausgewählt sein, lassen sich im Bereich Eigenschaften (**b**) die Eigenschaften des Diagramms (Name, Version und Parent) bearbeiten.

Sollte eine Node ausgewählt sein, werden stattdessen die Eigenschaften der Node eingeblendet. Die Eigenschaften unterscheiden sich hierbei je nachdem welche Art von Node ausgewählt wurde. So können zum Beispiel Eigenschaften wie die Position, die Größe, die Beschriftung und die Datentypen zur Verfügung stehen.

5.1.1.5 Menüleiste

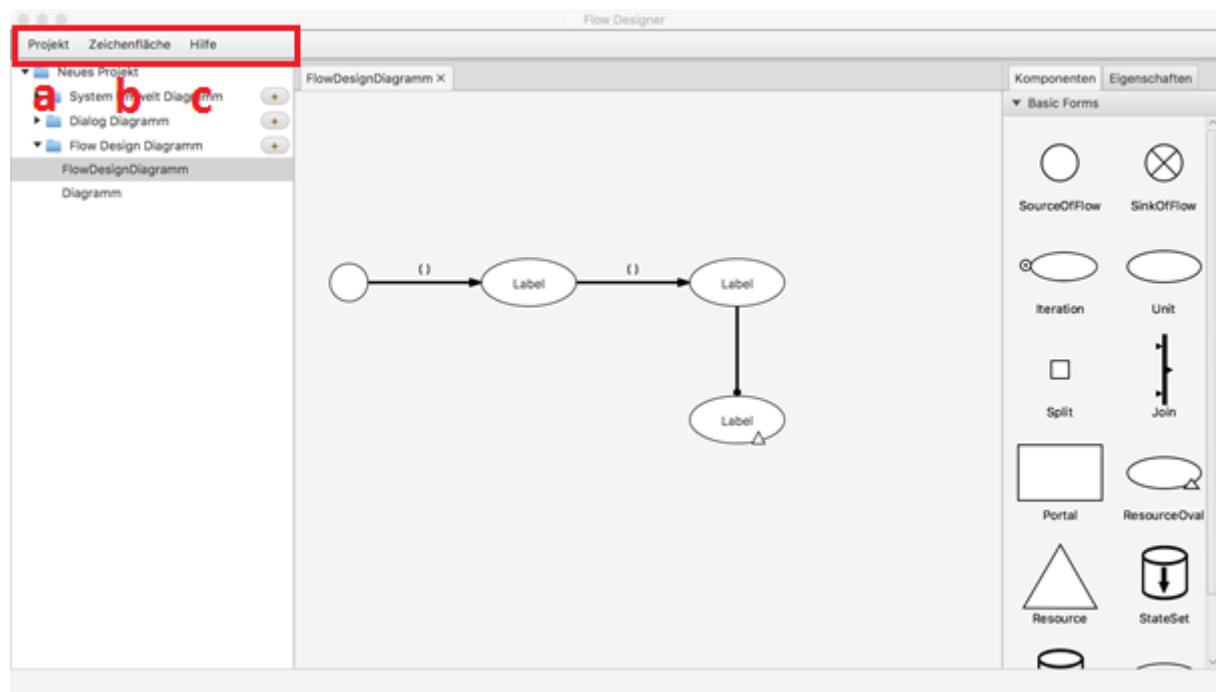


Abbildung 32 - Oberfläche: Menüleiste

Die Menüleiste enthält drei Menüpunkte.

Im Menüpunkt „Projekt“ (**a**) lässt sich ein neues Projekt erstellen, ein anderes Projekt öffnen, die Änderungen am geöffneten Projekt speichern und ebenfalls lässt sich hier das geöffnete Projekt schließen.

Unter dem Menüpunkt „Zeichenfläche“ (**b**) kann der Benutzer die Zoomstufe der Zeichenfläche speziell setzen. So kann man sich alle Elemente im Diagramm anzeigen lassen (Zoom-to-Fit) oder die Zoomstärke wieder auf 100% setzen. Ebenfalls gibt es hier die Option, alle anderen Bereiche außer der Zeichenfläche auszublenden und das Programm in den Vollbildmodus zu versetzen.

Im Letzen Menüpunkt „Hilfe“ (**c**) findet der Benutzer hilfreiche Informationen über die Nutzung der Software, wie zum Beispiel eine Liste der definierten Tastenkürzel um das Programm schneller zu bedienen.

6 Ausblick

In diesem Abschnitt wollen wir einige Möglichkeiten aufzeigen, wie die Software in Zukunft ausgebaut und verbessert werden kann. Wir stellen den Source Code unter der Open Source Lizenz zur Verfügung, wodurch es für spätere Projektgruppen möglich sein wird, an der Software weiter zu entwickeln.

6.1 Feature Vorschläge

6.1.1 Rückgängig

Um einen Fehler den man gemacht hat, oder einfach einen älteren Stand des Diagramms wieder herzustellen zu können, gibt es in anderen Programmen oftmals die Funktion Geschehenes rückgängig zu machen.

Ein Ansatz zur Realisierung eines Solchen Features wäre, die bereits eingebaute Versionierung zu verwenden. Dabei werden diese Versionen aber nicht auf der Festplatte gespeichert, sondern liegen nur zu Laufzeit vor. Dieser Ansatz bedeutet aber bei großen Diagrammen auch viele Daten, welche zusätzlich im Arbeitsspeicher liegen. Daher müsste man diese Funktion durch ein Limit der gespeicherten „internen“ Versionen begrenzen.

Ein anderer Ansatz ist, dass man sich die durchgeföhrte Aktion abspeichert. Wird zum Beispiel eine Node hinzugefügt, wird diese Aktion gespeichert, um sie hinterher wieder rückgängig machen zu können.

6.1.2 Raster in der Zeichenfläche

Um die Zeichenfläche noch benutzerfreundlicher zu gestalten, könnte man ein Raster über die gesamte Fläche legen. Dies kann z.B. ähnlich wie bei der Unreal Engine realisiert werden.

Zudem wäre es denkbar, dass sich Elemente an diesem Raster ausrichten können, um das Ausrichten zu vereinfachen.

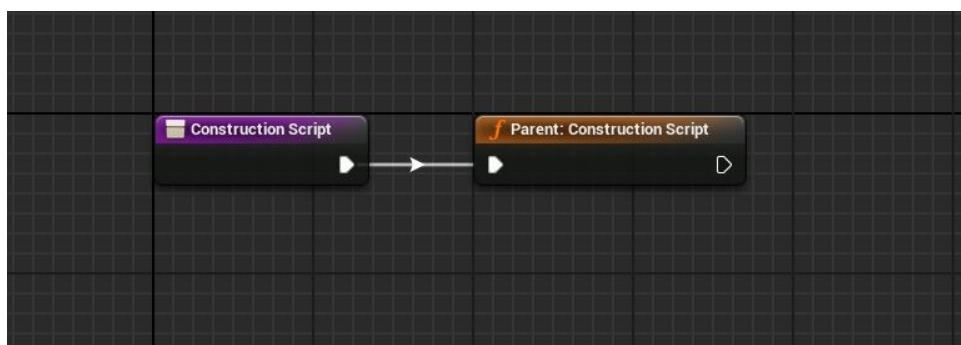


Abbildung 33 - Raster der Unreal Engine 4

6.1.3 Zeichenfläche immer verschiebbar

Eine weitere Möglichkeit die Nutzerfreundlichkeit der Zeichenfläche zu erhöhen, wäre wenn sich die Zeichenfläche in alle Richtungen verschieben lassen würde, auch wenn sich keine Elemente außerhalb des sichtbaren Bereiches befinden.

6.1.4 Zum Mauszeiger zoomen

Momentan wird beim Zoomen in der Zeichenfläche immer in die obere linke Ecke gezoomt. In diesem Fall würde es die Bedienung deutlich vereinfachen, wenn die Zeichenfläche immer zum Mauszeiger hin zoomen würde.

6.1.5 Beim Erstellen eines Diagramms den Parent angeben

Bei der Erstellung eines Diagramms kann zurzeit nur der Name angegeben werden. Zusätzlich wäre es denkbar, dass man auch direkt den Parent setzen kann, wodurch das Diagramm bereits hierarchisch richtig im Baumdiagramm erstellt werden würde.

Des Weiteren könnte man beim Rechtsklick auf ein Diagramm im Baumdiagramm die Möglichkeit schaffen, ein Diagramm als Kind-Diagramm zu erstellen.

6.1.6 Gruppierung

Um die Übersicht in einem Diagramm zu behalten, wäre eine Gruppierung von Units eine willkommene Möglichkeit. So wäre es möglich zum Beispiel Klassen, Namensräume oder Interfaces zu definieren. Momentan kann man dies nur mit dem Setzen der Hintergrundfarbe von Elementen realisieren.

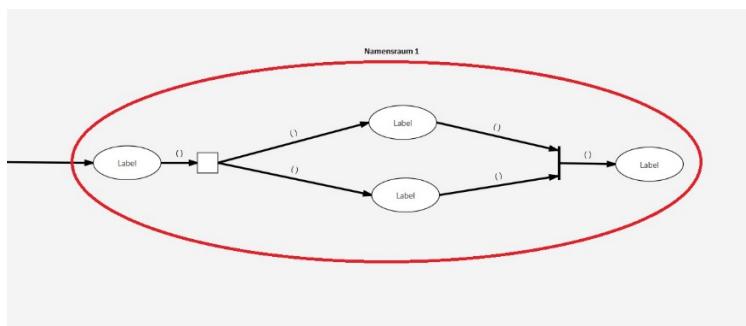


Abbildung 34 - Gruppierung von Elementen in einem Diagramm

6.1.7 Code Generierung

Eine Weiterentwicklung, welche nun als angegangen werden kann, ist die Generierung von Code aus den Diagrammen. Zu überlegen ist, wie man eine solche Funktion einbauen kann, welche mit verschiedenen Programmiersprachen zurechtkommt.

Das Ganze könnte aber auch anders herum funktionieren. Man generiert aus bestehendem Code Diagramme.

7 Fazit

Unser Ziel in dem Projekt „Flow Design“ war das Entwickeln eines intuitiven Werkzeuges zur Modellierung von Systemen nach Flow Design. Zu großen Teilen haben wir dieses Ziel und die damit zusammenhängenden Anforderungen erreicht.

7.1 Herausforderungen

Da zum Anfang des Projekts noch nicht geklärt war, wer die Betreuung unseres Teams übernehmen würde, verzögerte sich der Projektstart und wir erhielten erst eine Woche vor den Zwischenpräsentationen unser eigentliches Thema.

Herausfordernd war, dass die Meisten von uns zudem noch kaum Erfahrung beim Entwickeln von Software in einem Team hatten. Dadurch benötigte es auch einiges an Zeit, bis alle mit den eingesetzten Tools wie Git und Trello zurechtkamen.

Das Entwicklungskonzept Flow Design war uns zu diesem Zeitpunkt noch völlig unbekannt und die Recherche nach gültigen Notationen im Internet erwies sich als sehr zeitraubend. Als wir gerade fertig waren, eine Übersicht von allen Notationen zu erarbeiten, bekamen wir dann auch ein Cheet-Sheet über Flow Design von unserem Betreuer, welches die Arbeit deutlich erleichterte.

7.2 Erkenntnisse

Das Projekt half uns dabei, wichtige moderne Technologien, die bei der Entwicklung von Software eingesetzt werden, zu verstehen und auch anzuwenden. Hier ist unter anderem die Abhängigkeitsauflösung „Maven“, die Versionsverwaltung „Git“ und der Einsatz von „JavaFX“ für unser Frontend zu erwähnen.

Wir lernten den Aufbau von größerer Softwareprojekte kennen. Durch die Trennung des Codes in mehrere Module, wird uns ein strukturierter und übersichtlicher Aufbau der Software ermöglicht.

Es war auch eine großartige Möglichkeit, unser im Studium erlerntes Wissen, ganz praktisch anwenden zu können.

Angespornt von unserem Betreuer setzten wir uns tiefer mit den Prinzipien der Objektorientierten Programmierer auseinander. Zu nennen ist hier zum Beispiel das SOLID Prinzip, welches einige der wichtigsten Prinzipien zusammenfasst.

7.3 Danksagungen

Wir bedanken uns bei unserem Betreuer Herr Haybat und unserem Kunden Herr Erath für die gute Zusammenarbeit und dem Engagement, welches sie in das Projekt mit eingebracht haben.

Literaturverzeichnis

The Architects Napkin, 2016. Flow-Design Cheat Sheet – Part I, Notation [online] [Zugriff am: 18. Januar 2017]. Verfügbar unter:

<http://geekswithblogs.net/theArchitectsNapkin/archive/2011/03/19/flow-design-cheat-sheet-&ndash-part-i-notation.aspx>

Ralph Westphal, The Architect's Napkin – Ein Schummelzettel, 2014, ISBN 978-

1495312588 [online] [Zugriff am: 18. Januar 2017]. Verfügbar unter:

<https://leanpub.com/thearchitectsnapkin-derschummelzettel>

Ralf Westphal - Flow-Oriented Modelling, Youtube, [online] [Zugriff am: 18. Januar 2017]. <https://www.youtube.com/watch?v=I36OImYMIVs>

Cheat Sheet Flow Design, 2016. [online] [Zugriff am: 18. Januar 2017]. Verfügbar unter:

<http://refactoring-legacy-code.net/wp-content/uploads/2016/10/CheatSheet-Flow-Design.pdf>

IT-Designers Gruppe, 2016. [online] [Zugriff am: 18. Januar 2017]. Verfügbar unter:

<http://www.it-designers-gruppe.de/it-designers-gruppe/>