Labor für Informationstechnik und Produktionslogistik (LIP)
Verfahren, Strategien, Prozesse und IT-Systeme
Professor Dr.-Ing. Frank Herrmann

OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

Entwicklerhandbuch zur Losgrößen- und Ressourceneinsatzplanung bei Fließproduktion

Klassisches Losgrößenmodell, Mehrprodukproduktion

Juni 2015

Arnold Christiane
Denzin Timo
Eichinger Tobias
Sonnleitner Daniel
Wagner Pilar

Inhaltsverzeichnis

Inhaltsverzeichnis **I**

1 Einführung **1**
1.1 Aufbau des Dokuments 1
1.2 Problemstellung 2
1.3 Beschreibung der Software 2

2 Aufbau der Software **2**
2.1 Verwendete Techniken 2
2.2 Komponenten 5

3 Implementierung **9**
3.1 Verwendete Bibliotheken 9
3.2 JFreeChart 10
3.3 Zoomen 11
3.4 Algorithmen 11
3.4.1 Klassiche Losgrößenberechnung 11
3.4.2 Mehrproduktlosgrößen 13

4 Abkürzungsverzeichnis **15**

1 Einführung

Das vorliegende Dokument ist ein Handbuch für Entwickler des *"Static Multi-product Proportional Lotsizing and Scheduling Problem (SMPLSP)"*-Tools zur Losgrößen- und Ressourceneinsatzplanung bei Fließproduktion.
Die Software ist in Java 8 geschrieben und die Benutzeroberfläche wurde mittels JavaFX 8 realisiert.
Das folgende Bild zeigt die Software.

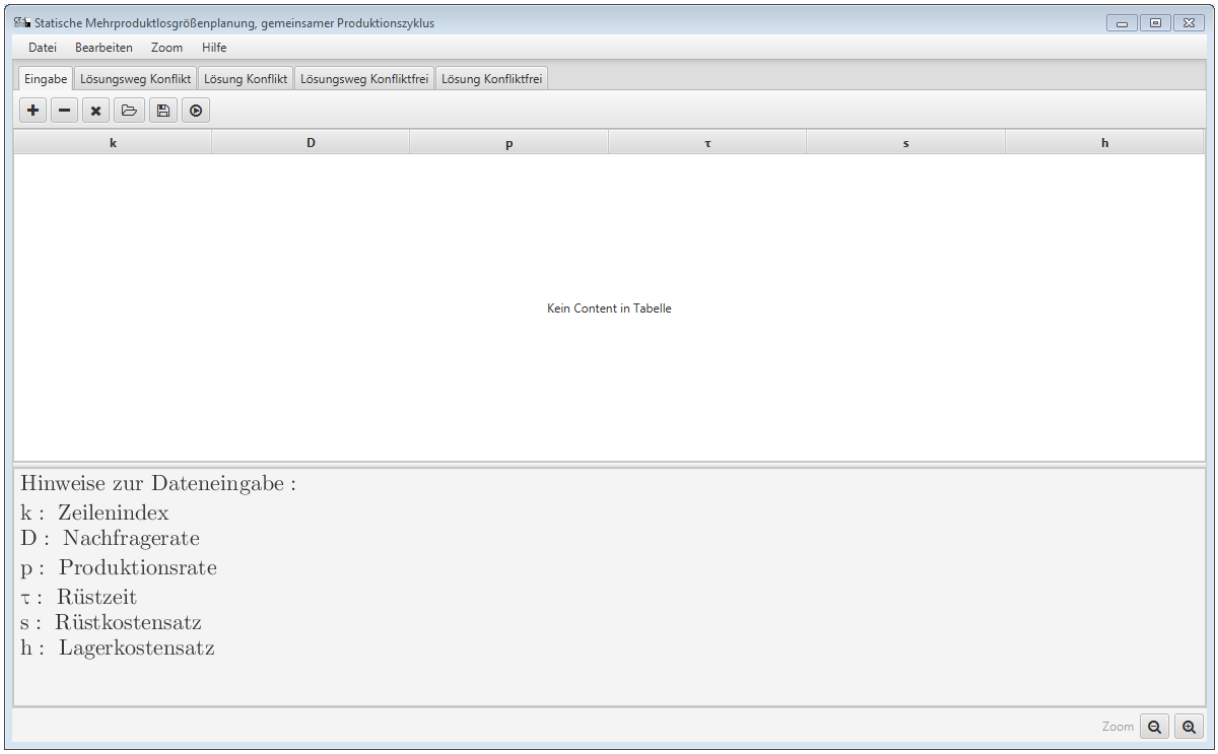

Abbildung 1: Oberfläche des SMPLSP-Tools

1.1 Aufbau des Dokuments

Zu Beginn soll die Problemstellung erklärt und der Ablauf des hier verwendeten Verfahrens erläutert werden.
Anschließend folgt eine kurze Beschreibung der Funktionen der Software.
Unter dem Punkt Aufbau der Software sollen die einzelnen Komponenten erläutert werden.
Im letzten Kapitel wird auf die Implementierung der einzelnen Komponenten eingegangen.

1.2 Problemstellung

Das vorliegende Programm dient zur Losgrößen- und Ressourceneinsatzplanung bei Fließproduktion. Dabei werden sowohl die Produktionszyklen für ein einzelnes Produkt, als auch der gemeinsame Zyklus für mehrere Produkte berechnet.

1.3 Beschreibung der Software

Nach dem Start des Tools müssen die Daten zur Berechnung eingegeben werden. Dies geschieht entweder manuell oder durch das Laden einer Datei. Es wird der Import von Comma-separated values (CSV)-Dateien unterstützt.
Nachdem die Produktionszyklen berechnet wurden, werden in den verschiedenen Reitern der Oberfläche die Ergebnisse in Form von Tabellen und Diagrammen dargestellt. Bei einem Klick in eine Tabellenzeile, erscheinen in der Erklärkomponente Details zur Berechnung dieses Wertes.
Detailliertere Information über die Funktionen der Software finden Sie im Benutzerhandbuch.

2 Aufbau der Software

Im Folgenden werden die verwendete Technik und die Komponenten des Programms vorgestellt. Die Komponenten werden durch Packages realisiert, daher werden in dieser Dokumentation auch die Packagenamen verwendet. Im weiteren Verlauf werden die Begriffe Komponente und Package gleichbedeutend benutzt.

2.1 Verwendete Techniken

JavaFX 8 Die Benutzeroberfläche der Software wurde in JavaFX 8 entwickelt. JavaFX verwendet das Model-View-Controller Prinzip.
Um die Gestaltung des Graphical User Interface (GUI) zu vereinfachen, wird der SceneBuilder verwendet. Der SceneBuilder ist ein Tool zur einfachen Erstellung von JavaFX Layouts. Dabei muss kein Code geschrieben werden, da dies vom SceneBuilder übernommen wird. Das folgende Bild zeigt den SceneBuilder.

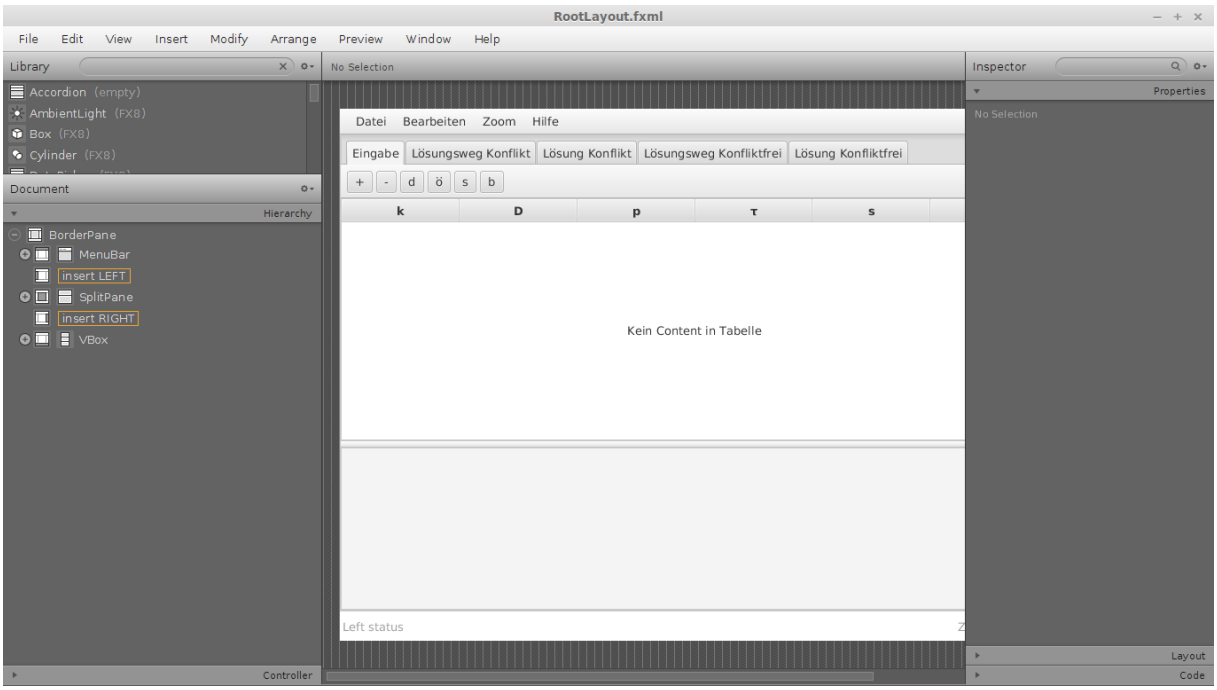

Abbildung 2: Der SceneBuilder

Beim Model-View-Controller Prinzip enthält die View lediglich den grafischen Aufbau der Oberfläche. Dieser Aufbau wird einer speziellen Extensible Markup Language (XML) Datei gespeichert, einer sogenannten FXML Datei.
Eine zugehörige Controller Klasse implementiert die, zur View gehörige, Logik in einer Java Klasse.
Im Model werden die verwendeten Daten abgespeichert und verwaltet.
Weder Model noch View enthalten Logik. Dies führt zu einer sauberen Schichtentrennung zwischen Benutzeroberfläche (View), Logik (Controller) und Datenhaltung (Model).

In der FXML-Datei wird die Controller Klasse gesetzt, dadurch kann der Compiler, beim Übersetzten des Programms, Controller und View einander zuordnen. Im Folgenden ist zu sehen, wie dies im SceneBuilder umgesetzt werden kann.

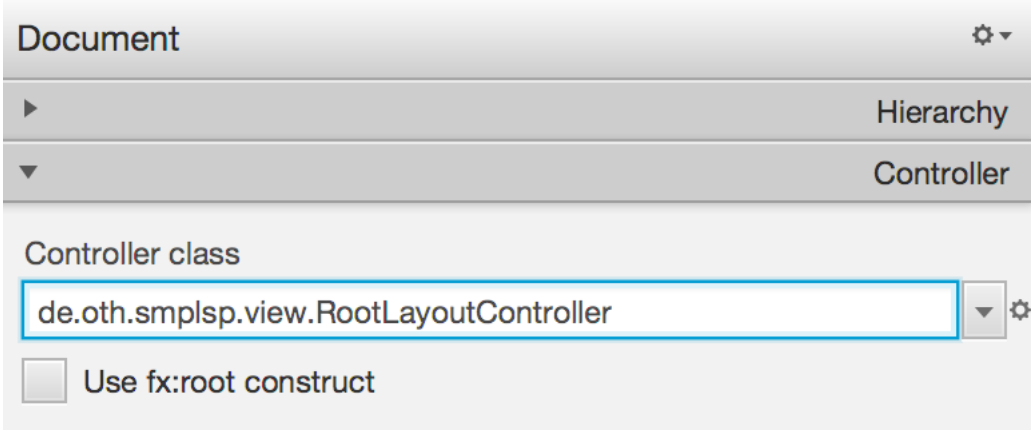

Abbildung 3: Controller Klasse im SceneBuilder setzten

Um auf die Elemente der GUI vom Controller aus zuzugreifen, werden beim Deklarieren eines Elements die @FXML Annotation gesetzt. Zusätzlich wird in der FXML-Datei die fx:id des Elements vergeben. Diese muss mit der Bezeichnung im Controller übereinstimmen.
In der folgenden Abbildung ist zu sehen, wie im SceneBuilder die fx:id gesetzt wird.

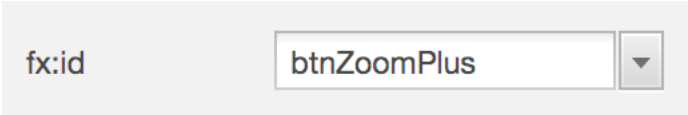

Abbildung 4: Setzten der fx:id im SceneBuilder

Und hier das Beispiel im Code:

```
1    // Code ...
2
3    @FXML
4    private Button btnZoomPlus;
5
6    // Code...
```

Listing 1: Beispielprogramm

Weitere Informationen finden Sie in diesem Tutorial zu JavaFX.
(http://code.makery.ch/library/javafx-8-tutorial/)

JFreeChart Zur Visualisierung der Losgrößen wird das Java-Framework JFreeChart verwendet. Mit JFreeChart können verschiedene Diagramme erstellt werden, darunter auch die hier verwendeten Gantt-Diagramme.

JUnit Um die implementierte Logik automatisiert testen zu können, werden JUnit - Tests verwendet. Dabei wird eine eigene Testklasse geschrieben, die feste Eingabedaten enthält. Anschließend wird die zu testende Methode ausgeführt und die Rückgabewerte der Methode automatisiert mit den erwarteten Ausgabedaten verglichen. Stimmen sie überein, war der Test erfolgreich.
Dies ermöglicht vor allem bei größeren Methoden ein Testen ohne Zutun des Entwicklers.

2.2 Komponenten

algorithms Hier sind die Algorithmen zur Berechnung der Losgrößen implementiert. Es gibt jeweils eine Klasse zur Berechnung der klassischen Losgrößen und zur Berechnung der Mehrproduktlosgrößen. Beide Klassen implementierten ein Interface, das gemeinsame Methoden vorgibt.
Der Produktionsprozess wird in beiden Fällen gleich berechnet, deshalb wird hierfür die selbe Klasse verwendet.

formula Die Komponente formula enthält die Klassen zur Erzeugung der Formeln, die in der Erklärkomponente angezeigt werden sollen. Die hierbei erzeugten Strings enthalten die mathematischen Formeln, in Form von LaTeX- Code. Diese werden dynamisch zur Laufzeit erzeugt und mit den passenden Daten gefüllt. Damit ist es möglich, zu jedem Ergebnis eine Formel mit eingesetzten Werten zu erzeugen.
Für die Formeln der klassischen Losgrößen und der Mehrproduktlosgrößen ist jeweils eine eigene Klasse implementiert. Außerdem ist noch eine Klasse für die Formeln des Produktionsprozesses und für produktspezifische Formeln vorhanden.

messages Dieses Package verwaltet die angezeigten Texte und Beschriftungen der Software. Dabei wird ein Ressource Handler verwendet, der es zusätzlich ermöglicht, das Programm in eine andere Sprache zu übersetzen, ohne große Änderungen am Code vorzunehmen.

persistence Hier werden die Dateizugriffe verwaltet. Hierfür wird für jeden Dateityp eine eigene Klasse erstellt, die von der Klasse AbstractFile erbt. Dadurch müssen Funktionen wie das Laden der Datei nicht erneut implementiert werden.

util In dieser Komponente werden, neben einigen Hilfsklassen, vor allem die Einstellungen des Tools verwaltet. Hierfür wurde eine Singleton Klasse Configuration geschaffen.

zoom In der Komponente zoom sind die Klassen zum Vergrößern und Verkleinern der Benutzeroberfläche enthalten. Dies wird mittels Veränderungen in den Cascading Style Sheets (CSS) realisiert.

error Hier werden die Exceptions und andere Fehlermeldungen verwaltet.

model Die Model - Klassen zur Datenhaltung werden in diesem Package aufbewahrt. Dazu gehören die Abstraktionen von Produkt, Produktionsprozess und dem Ergebnis der Analyse.

test Die Komponente test enthält die Klassen zur Durchführung der JUnit - Tests.

view In dieser Komponente befinden sich die Controller und FXML Dateien der grafischen Oberfläche. Diese ist in ein RootLayout und die fünf Reiter unterteilt. Das RootLayout definiert hier den Rahmen für die Reiter, die Menüleiste und die Unterleiste der Applikation.
Die Reiter sind in das RootLayout integriert, besitzen aber eigene FXML-Dateien und Controller-Klassen. Dies wird im SceneBuilder wie folgt dargestellt:

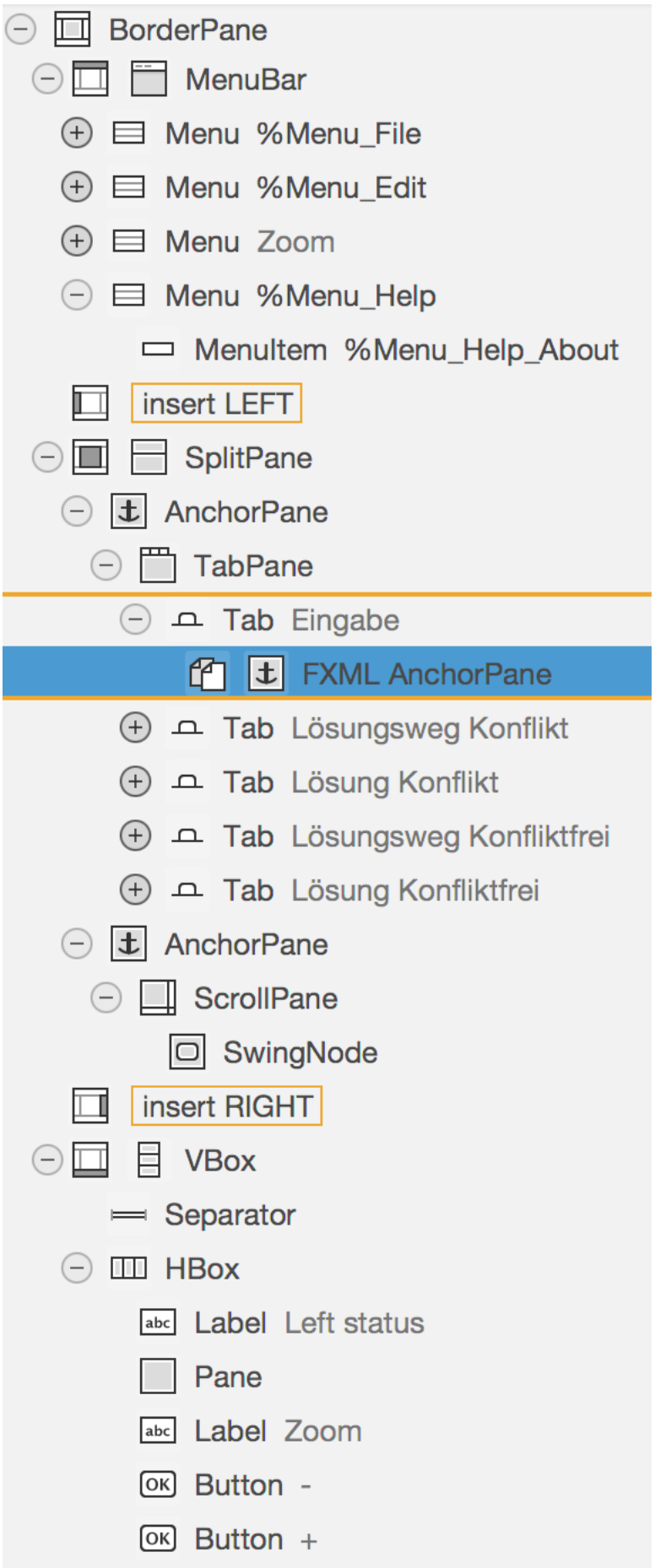

Abbildung 5: Darstellung der Reiter im SceneBuilder

Damit die Unterkomponenten des RootLayout miteinander kommunizieren und aufeinander zugreifen können, wird ihnen der RootLayoutController übergeben. Dadurch ist es ihnen möglich auf die Elemente und Teile des RootLayout zuzugreifen.

Nachfolgend sind die einzelnen Komponenten im Package - Explorer zu sehen.

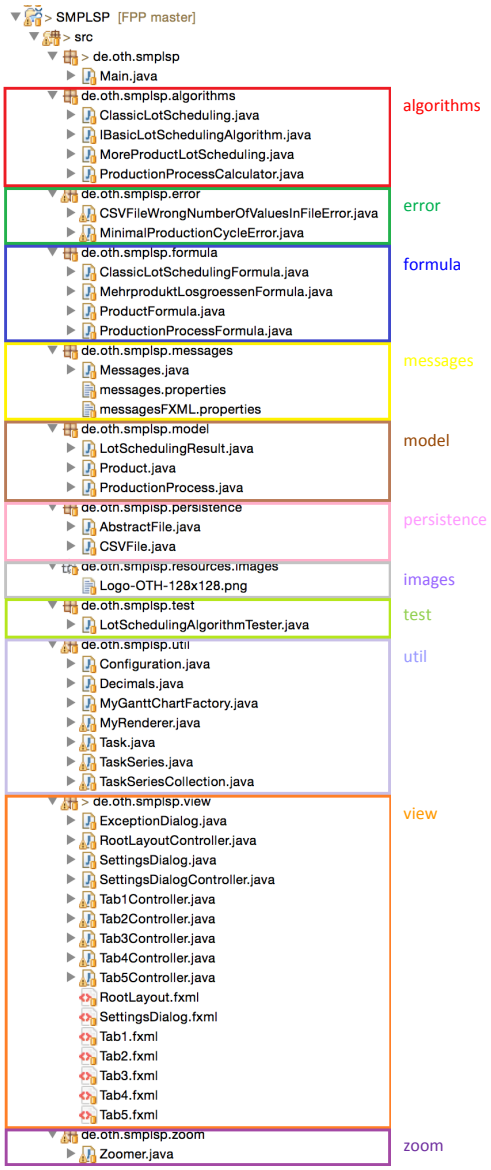

Abbildung 6: Komponenten im Package - Explorer

3 Implementierung

Dieses Kapitel beschreibt die Implementierung, sowie die bei der Entwicklung der Komponenten verwendeten Bibliotheken. Dabei wird vor allem auf die Algorithmen der Losgrößenberechnung eingegangen. Die einzelnen Klassen werden in der Javadoc beschrieben.

3.1 Verwendete Bibliotheken

Die verwendeten Bibliotheken sind im Ordner lib gespeichert, um einen pfadunabhängigen Import in den Java Build Path zu ermöglichen.
Sie unterteilen sich in fünf Einsatzbereiche:

CSV-Import: Diese Bibliotheken bieten Klassen zum automatisierten Einlesen und Beschreiben von CSV-Dateien an. Dies wird mittels einer Parser und einer Printer Klasse erreicht.

JavaFX Zusätze: Die hier verwendeten Bibliotheken erweitern JavaFX um vorgefertigte Dialoge und Buttons.

JFreeChart: Das Einbinden dieser Bibliotheken ermöglicht es Daten mittels Diagrammen zu visualisieren. Dabei wird die Klasse JFreeChart mit Daten gefüllt und dann zur Anzeige gebracht.

LaTeX: Die Formeln in der Erklärkomponente werden mittels LaTeX- Code erzeugt. Der zur Verfügung gestellten Klasse TeXFormula wird der LaTeX- Code übergeben und daraus ein Bild generiert.

Resource Handler: Der Resource Handler ersetzt die Eclipse eigene Funktion Source → Externalize Strings. Diese dient zur Auslagerung der Strings im Java Code in eine eigene Properties Datei. Dies hat den Vorteil, dass der Code übersichtlicher wird und die Texte in mehrere Sprachen übersetzt werden können, ohne in den Klassen etwas zu ändern.

Nachfolgende Abbildung zeigt die verwendeten Bibliotheken in der Ordnerstruktur.

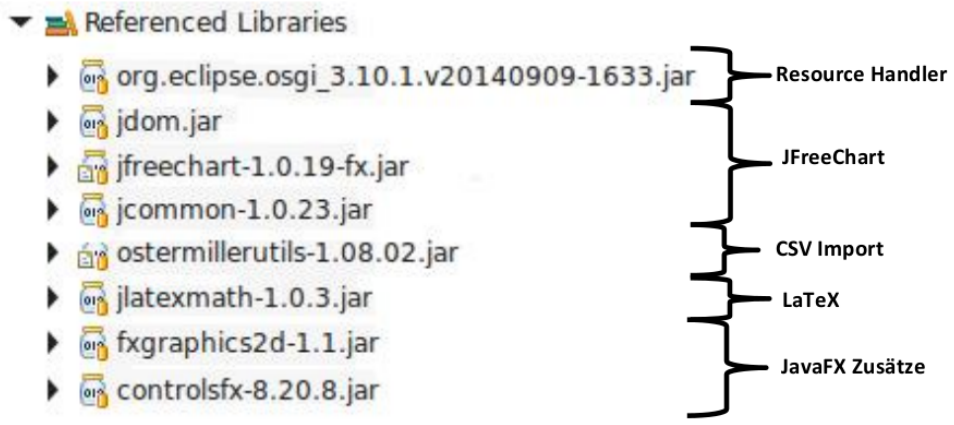

Abbildung 7: Verwendete Bibliotheken

3.2 JFreeChart

Die Diagramme in Reiter 3 und Reiter 5 wurden mit JFreeChart realisiert.
Da die übergebenen Start- und Endwerte der einzelnen Balken Gleitkommazahlen sind, JfreeChart aber standardmäßig nur mit Date- oder Integerwerten umgehen kann, mussten einige JfreeChart Klassen angepasst werden. Diese veränderten Klassen sind im package util zu finden. Nachfolgende Abbildung zeigt dieses im Packageexplorer.

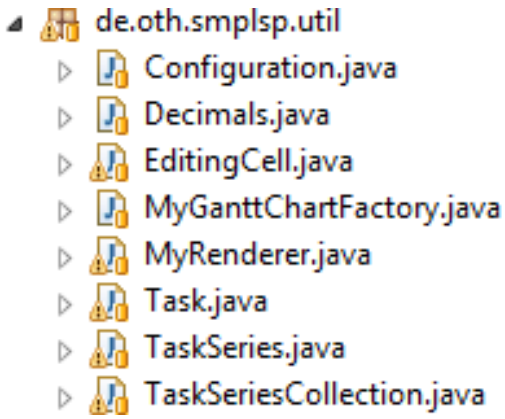

Abbildung 8: Veränderte Klassen für JFreechart

Die Klassen Task, TaskSeries und TaskSeriesCollection wurden verändert, um mit Gleitkommazahlen umgehen zu können.
Die Klasse MyRenderer übernimmt die richtige Farbgebung der Balken innerhalb der Charts. In MyGanttChartFactory werden die Achsen beschriftet, die Legende des Charts erzeugt und die Hintergrundfarbe festgelegt.
Sämtliche benötigten Methoden und Klassen werden aus dem Controller der jeweiligen

View aufgerufen. Der Methode createDataset wird die Liste der Produktionsprozesse aus dem Controller übergeben und daraus eine TaskSeriesCollection erzeugt. Wichtig hierbei ist, dass jede Zeile einen Task darstellt und die Balken für Rüst- und Produktionszeit in einem Subtask abgebildet werden. Alle Subtasks werden dann dem übergeordneten Task hinzugefügt. Abschließend werden alle Tasks zu einer TaskSeries hinzugefügt und diese in eine TaskSeriesCollection aufgenommen. Diese endgültige TaskSeriesCollection wird nun mittels showChart und createChart an die oben beschriebene MyGanttChartFactory übergeben. Diese erstellt daraus ein Gantt Chart und zeigt dieses in der Benutzeroberfläche an.

3.3 Zoomen

Die Zoomfunktion wird wird über die Klasse Zoomer im Package zoom gesteuert. Dort sind verschiedene Schriftgrößen definiert (Schriftgröße für Standard-Fonts, Schriftgröße für die Latex-Fonts (Erklärkomponente) und Schriftgröße für das JFreeChart).
Die verschiedenen Komponenten der Oberfläche werden wie folgt skaliert:

- Für die verwendeten Icon-Fonts werden die Icons in der neuen Schiftgröße neu berechnet.
- Bei den JFreeChart Diagrammen wird analog verfahren.
- Der Inhalt der Erklärkomponente wird ebenfalls mit der neuen Schriftgröße neu berechnet.
- Für die restlichen GUI-Elemente erfolgt eine Modifikation des verwendeten CSS-Styles. Hierbei wird die CSS-Style-fx-font-size mit der neuen Schriftgröße gesetzt und geladen.

3.4 Algorithmen

In diesem Programm werden zwei verschiedene Verfahren verwendet. Die klassische Losgrößenberechnung und die Mehrproduktlosgrößenberechnung. Zur Verdeutlichung wird Pseudocode verwendet.

3.4.1 Klassiche Losgrößenberechnung

Zu Beginn findet die Losgrößenberechnung für ein Produkt statt. Diese ist in fünf Schritte unterteilt:

1. Berechne die Lose für jedes Produkt

2. Berechne die Produktionszeit für jedes Produkt
3. Berechne die Effizienz der Maschine
4. Berechne den optimalen Produktionszyklus für jedes Produkt
5. Berechne die Reichweite für die einzelnen Produkte

Nachfolgend ist der Ablauf in Pseudocode dargestellt.

```
classicLotScheduling(List<Product> products){

    Map<Integer, Double> tOptSingle = new HashMap<Integer, Double>();
    LotSchedulingResult result;

  // calculateBatchSize 1
  for (Product product : products) {
      product.setQ(Math.sqrt((2 * product.getD() * product.getS())
        / (product.getH() * (1 - (product.getD() / product.getP())))));
  }
  // calculateProductionTime 2
  for (Product product : products) {
      product.setT(product.getQ() / product.getP());
  }
  // calculateEfficiencyOfMachine 3
  for (Product product : products) {
      product.setRoh(product.getD() / product.getP());
  }
  // calculateOptProductionCycle 4
  for (Product product : products) {
      tOptSingle.put(
        product.getK(),
        Math.sqrt((2 * product.getS() / (product.getH()
  }
  // calculateRange 5
  for (Product product : products) {
      product.setR(product.getQ() / product.getD());
  }
  return new LotSchedulingResult(products, tOptSingle);
}
```

Listing 2: Beispielprogramm

3.4.2 Mehrproduktlosgrößen

Im zweiten Durchlauf findet die Losgrößenberechnung für mehrere Produkt statt. Diese ist in sechs Schritte unterteilt:

1. Berechne die Effizienz der Maschine
2. Berechne den Produktionszyklus für die Produkte
3. Berechne den minimalen Produktionszyklus
4. Berechne den optimalen gemeinsamen Produktionszyklus
5. Berechne die Produktionszeit für jedes Produkt
6. Berechne die Reichweite für die einzelnen Produkte

Nachfolgend ist der Ablauf in Pseudocode dargestellt.

```
1  MoreProductLotScheduling(List<Product> products){
2
3      LotSchedulingResult result;
4      Double tOpt;
5    Double tMin;
6
7    // calculateEfficiencyOfMachine 1
8    for (Product product : products) {
9        product.setRoh(product.getD() / product.getP());
10   }
11
12   // calculateOptProductionCycle 2
13   double numerator = 0.0;
14   double denominator = 0.0;
15
16   for (Product product : products) {
17       numerator += product.getS();
18   }
19
20   numerator *= 2;
21
22   for (Product product : products) {
23       denominator += (product.getH() * product.getD() * (1 - product
24           .getRoh()));
25   }
26
27   tOpt = Math.sqrt(numerator / denominator);
28
29   // calculateMinProductionCycle 3
```

```
30      double numerator = 0.0;
31      double denominator = 0.0;
32
33      for (Product product : products) {
34          numerator += product.getTau();
35      }
36
37      for (Product product : products) {
38          denominator += product.getRoh();
39      }
40
41      denominator = 1 - denominator;
42
43      tMin = (numerator / denominator);
44
45      if (tOpt < tMin) {
46          throw new MinimalProductionCycleError();
47      }
48
49      // calculateBatchSize 4
50      for (Product product : products) {
51          product.setQ(product.getD() * tOpt);
52      }
53
54      // calculateProductionTime 5
55      for (Product product : products) {
56          product.setT(product.getQ() / product.getP());
57      }
58
59      // calculateRange 6
60      for (Product product : products) {
61          product.setR(product.getQ() / product.getD());
62      }
63
64      return new LotSchedulingResult(products, tOpt, tMin);
65  }
```

Listing 3: Beispielprogramm

4 Abkürzungsverzeichnis

CSS Cascading Style Sheets.

CSV Comma-separated values.

GUI Graphical User Interface.

SMPLSP Static Multi-product Proportional Lotsizing and Scheduling Problem.

XML Extensible Markup Language.