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

1 Einführung

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

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 beschrieben und deren Zusammenspiel erläutert werden.
Im letzten Kapitel wird auf die Implementierung der einzelnen Komponenten eingegangen.

1.2 Problemstellung

Das vorliegende Programm, dient zur Lösungen von, Losgrößen- und Ressourceneinsatzplanung bei Fließproduktion. Dabei werden die 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, diese können entweder manuell eingegeben werden, oder von einer Datei geladen werden. Momentan wird nur der Import von Comma-separated values (CSV)-Dateien unterstützt.
Nachdem die Produktionszyklen berechnet wurden, werden in den verschiedenen Tabs, die Ergebnisse in Form von Tabellen und Diagrammen dargestellt. Bei einem Klick in eine Zeile der Tabellen, erscheint in der Erklärkomponente die Berechnung dieses Wertes.
Detailliertere Information über die Funktionen der Software finden Sie im Benutzerhandbuch.

2 Aufbau der Software

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

2.1 Verwendete Techniken

JavaFX 8 Die Benutzeroberfläche der Software wurde in JavaFX 8 entwickelt. JavaFX vewendet das Model-View-Controller Prinzip. Bei diesem Aufbau enthält die View lediglich den grafischen Aufbau der Oberfläche. Dieser Aufbau wird ein einer speziellen Extensible Markup Language (XML) Dateien gespeichert, sogenannten FXML Dateien. 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).

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 Ein- und Ausgabedaten enthält. Dann wird die zu testende Methode ausgeführt und die Rückgabewerte der Methode mit den erwarteten Ausgabedaten verglichen. Stimmen sie überein, war der Test erfolgreich.
Dies ermöglicht vor allem bei größeren Methoden ein automatisiertes 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 Formeln, in Form von LaTeXCode. 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 ebenfalls ermöglicht, das Programm in eine andere Sprache zu übersetzen, ohne große Änderungen am Code vorzunehmen.

persistence Hier werden die die Dateizugriffe verwaltet. Hierfür wird für jeden Dateityp eine eigene Klasse erstellt, die von der Klasse AbstractFile erben. Somit 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, die die Einstellungen der Software verwaltet.

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) Style realisiert.

error Hier werden die Exceptions und andere Fehlermeldungen verwaltet.

model Die Model - Klassen zur Datenerhaltung 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 - Test.

view In dieser Komponenten befinden sich die Controller und FXML Dateien der grafischen Oberfläche. Diese ist dabei in ein RootLayout und die fünf Tabs unterteilt. Das RootLayout definiert hier den Rahmen für die Tabs, die Menüleiste und die Unterleiste der Applikation. Die Tabs sind in das RootLayout integriert, besitzen aber eigene FXML-Dateien und Controller-Klassen.

3 Implementierung

Dieses Kapitel beschreibt die Implementierung und die bei der Entwicklung der Komponente verwendeten Bibliotheken. Dabei wird auf vor allem auf die Algorithmen der Losgrößenberechnung eingegangen.

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 Printer Klasse erreicht.

JavaFX: Um JavaFX als grafische Oberfläche zu nutzten, sind einige zusätzliche Bibliotheken nötig. Außerdem sind auch Codebeispiele und vorgefertigte Dialoge enthalten.

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 LaTeXCode erzeugt. Der zur Verfügung gestellten Klasse TeXFormula wird der LaTeXCode 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 es außerdem die Texte in mehreren Sprachen vorliegen können, ohne in den Klassen etwas zu ändern.

Nachfolgende Abbildung zeigt die verwendeten Bibliotheken in der Ordnerstruktur.

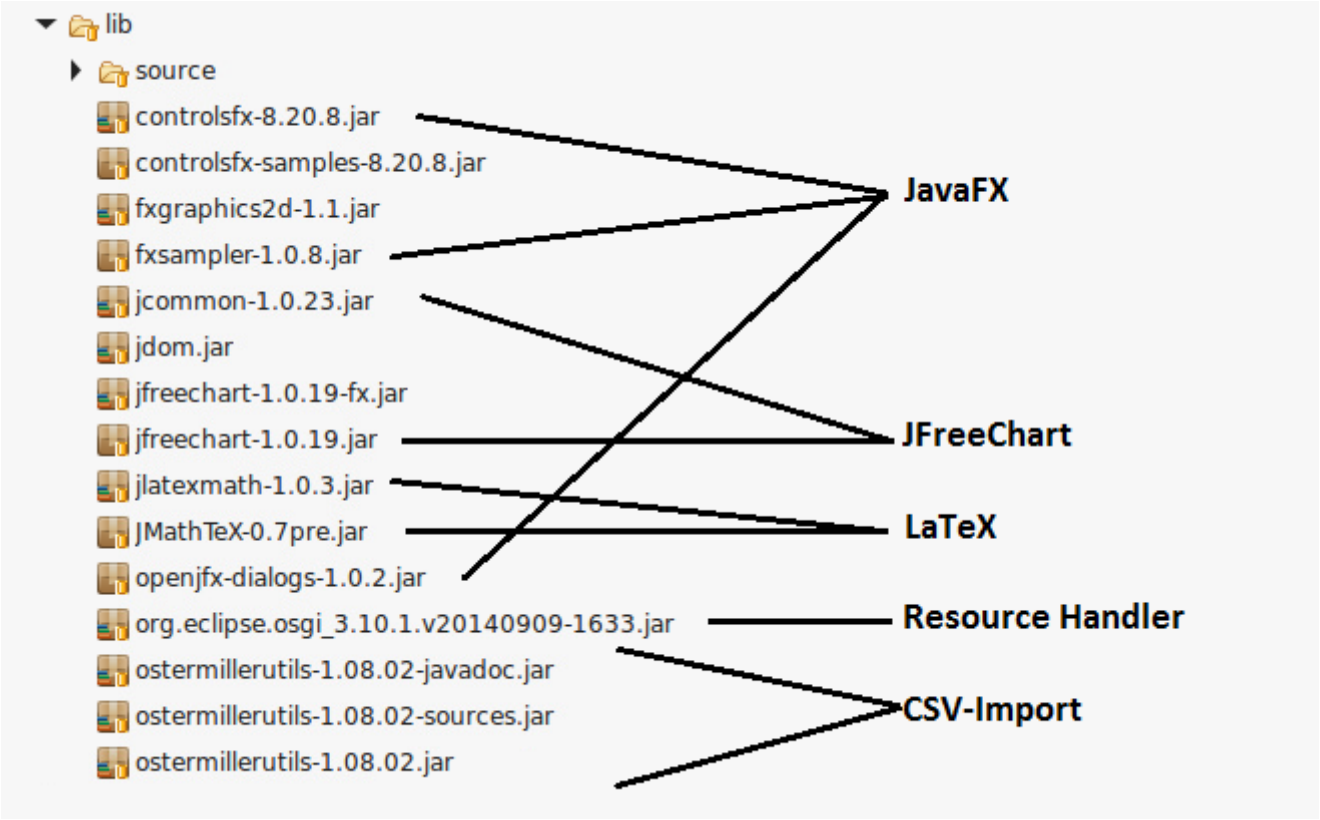

Abbildung 1: Verwendete Bibliotheken

3.2 Algorithmen

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

3.2.1 Klassiche Losgrößenberechnung

Zu Beginn wird 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.

```
1  classicLotScheduling(List<Product> products){
2
3      Map<Integer, Double> tOptSingle = new HashMap<Integer, Double>();
4      LotSchedulingResult result;
5
6    // calculateBatchSize 1
7    for (Product product : products) {
8        product.setQ(Math.sqrt((2 * product.getD() * product.getS())
9          / (product.getH() * (1 - (product.getD() / product.getP())))));
10   }
11   // calculateProductionTime 2
12   for (Product product : products) {
13       product.setT(product.getQ() / product.getP());
14   }
15   // calculateEfficiencyOfMachine 3
16   for (Product product : products) {
17       product.setRoh(product.getD() / product.getP());
18   }
19   // calculateOptProductionCycle 4
20   for (Product product : products) {
21       tOptSingle.put(
22         product.getK(),
23         Math.sqrt((2 * product.getS() / (product.getH()
24   }
25   // calculateRange 5
26   for (Product product : products) {
27       product.setR(product.getQ() / product.getD());
28   }
29   return new LotSchedulingResult(products, tOptSingle);
30 }
```

Listing 1: Beispielprogramm

3.2.2 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 2: Beispielprogramm

4 Abkürzungsverzeichnis