Labor für Informationstechnik und Produktionslogistik (LIP)

Verfahren, Strategien, Prozesse und IT-Systeme

Professor Dr.-Ing. Frank Herrmann

OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

Entwicklerhandbuch für die Themen

Capacitated Lot-Sizing Problem

und

Hauptproduktionsprogrammplanung

Februar 2015

Arnold Christiane

Butz Thomas

Denzin Timo

Eichinger Tobias

Gais Dominik

Liebich Johannes

Schertler Sascha

Sonnleitner Daniel

Wagner Pilar

Inhaltsverzeichnis

1 Einführung

Das vorliegende Dokument ist ein Handbuch für Entwickler des *"Hauptproduktionsprogrammplanung (HPPLAN)"*-Tools zur Produktionsprogrammplanung im Rahmen der operativen Planung und des Tools *"Capacitated Lot-Sizing Problem (CLSP)"* zur dynamischen Losgrößenplanung.

Die Software ist in Java 8 geschrieben und die Benutzeroberfläche wurde mittels JavaFX 8 realisiert.

Nachfolgende Bilder zeigen die Oberflächen der Programme.

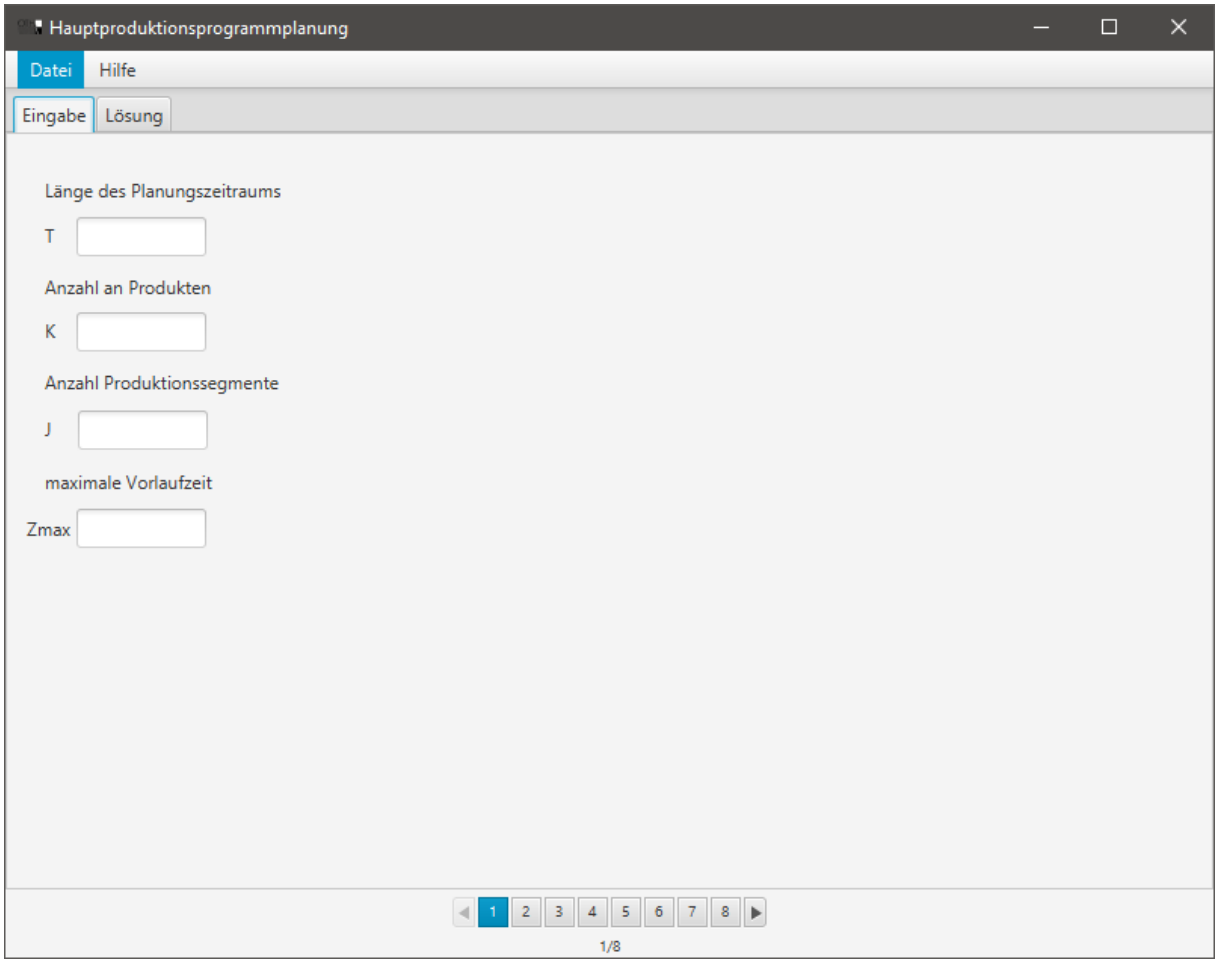

Abbildung 1: Oberfläche des HPPLAN-Tools

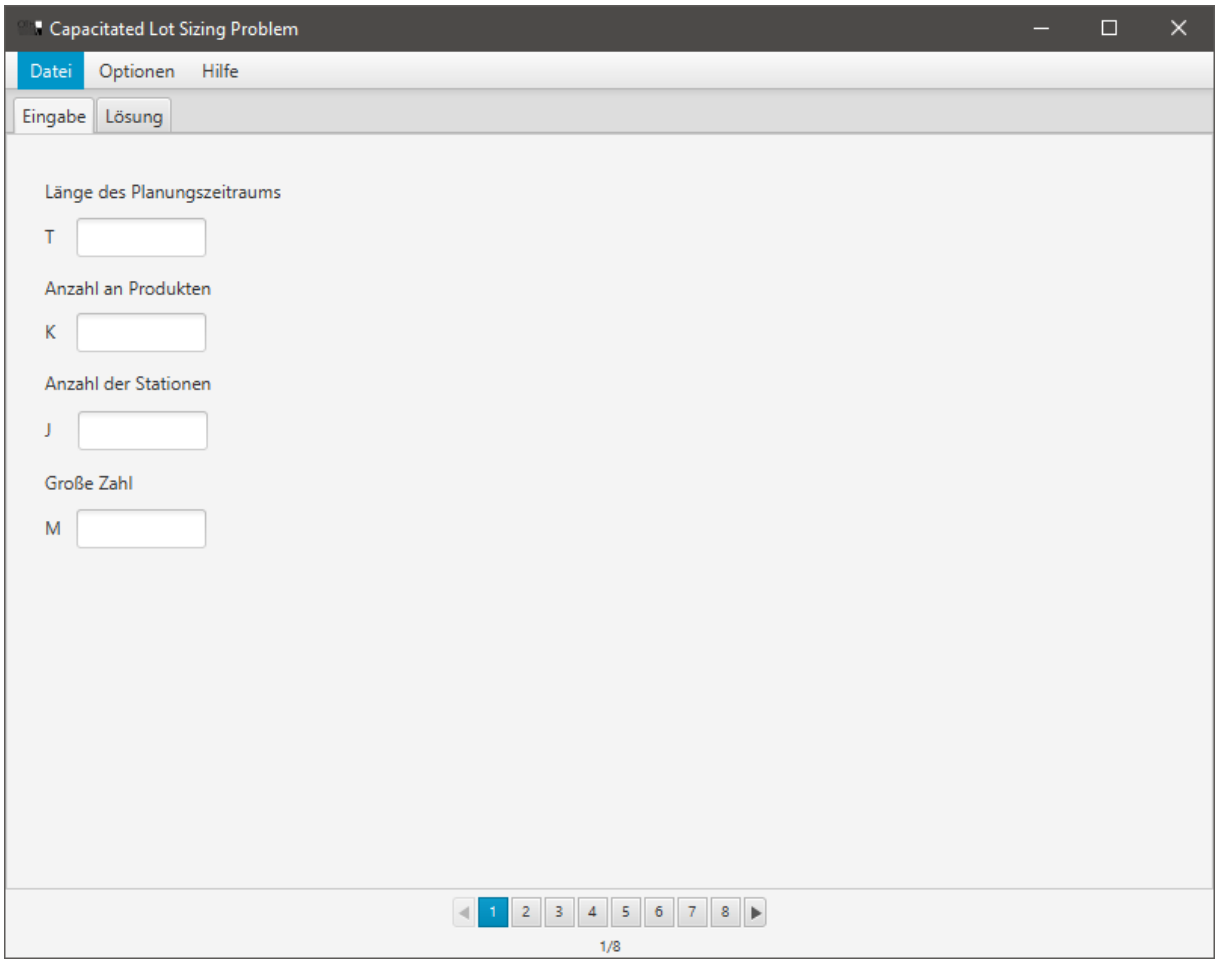

Abbildung 2: Oberfläche des CLSP-Tools

1.1 Aufbau des Dokuments

Zu Beginn sollen die Problemstellungen der beiden Themen erklärt und die Abläufe der hier verwendeten Verfahren 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 HPPLAN

Das vorliegende HPPLAN-Tool dient zur Lösung der Hauptproduktionsprogrammplanung. Dabei soll berechnet werden, welche Mengen der einzelnen Produkte in welcher Periode und in welchem Produktionssegment produziert werden sollen. Die Berechnet erfolgt mit dem ILOG-Framework.

1.3 Problemstellung CLSP

CLSP ist ein Modell der dynamischen Losgrößenplanung. Es geht dabei von mehreren Produkten aus für die eine begrenzte Produktionskapazität vorhanden ist. Zu ermitteln ist in welcher Periode welche Lose aufgelegt werden sollen und wie groß diese sein sollen. Die Bedarfe der einzelnen Perioden werden dabei als bekannt angenommen. Die CLSP-Berechnung erfolgt ebenfalls mit dem ILOG-Framework.

1.4 Beschreibung der Software

Nach dem Start des Tools müssen die Daten zur Berechnung eingegeben werden. Dies geschieht entweder manuell oder durch das Laden einer mit den notwendigen Informationen gefüllten DAT-Datei. Die manuelle Eingabe der Daten ist auf mehrere Seiten aufgeteilt, um die Übersicht zu erhöhen.
Die übergebenen Daten werden in einem temporären Modell gespeichert, an das ILOG-Framework übergeben und in diesem berechnet. Die Ausgabe des Frameworks wird entgegengenommen und in dem Tab "Lösungäusgegeben.
Detailliertere Information über die Funktionen der beiden Tools finden Sie in den Benutzerhandbüchern.

2 Aufbau der Software

Im Folgenden werden die verwendete Technik und die Komponenten der Programme 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.
Unter dem Java 7 SDK ist die notwendige Bibliothek "jfxrt.jar"noch nicht im Classpath enthalten. Deshalb muss diese zuvor eingebunden werden.

1. Einstellungen des Projekts in Eclipse öffnen
2. Im Reiter Libraries "Add External JARs" wählen

3. Unter [Pfad zum JDK]\jdk1.7.[0_06 oder höher] \jre\lib findet man die jfxrt.jar

Weitere Informationen unter http://blog.essential-bytes.de/javafx_in_eclipse/

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.

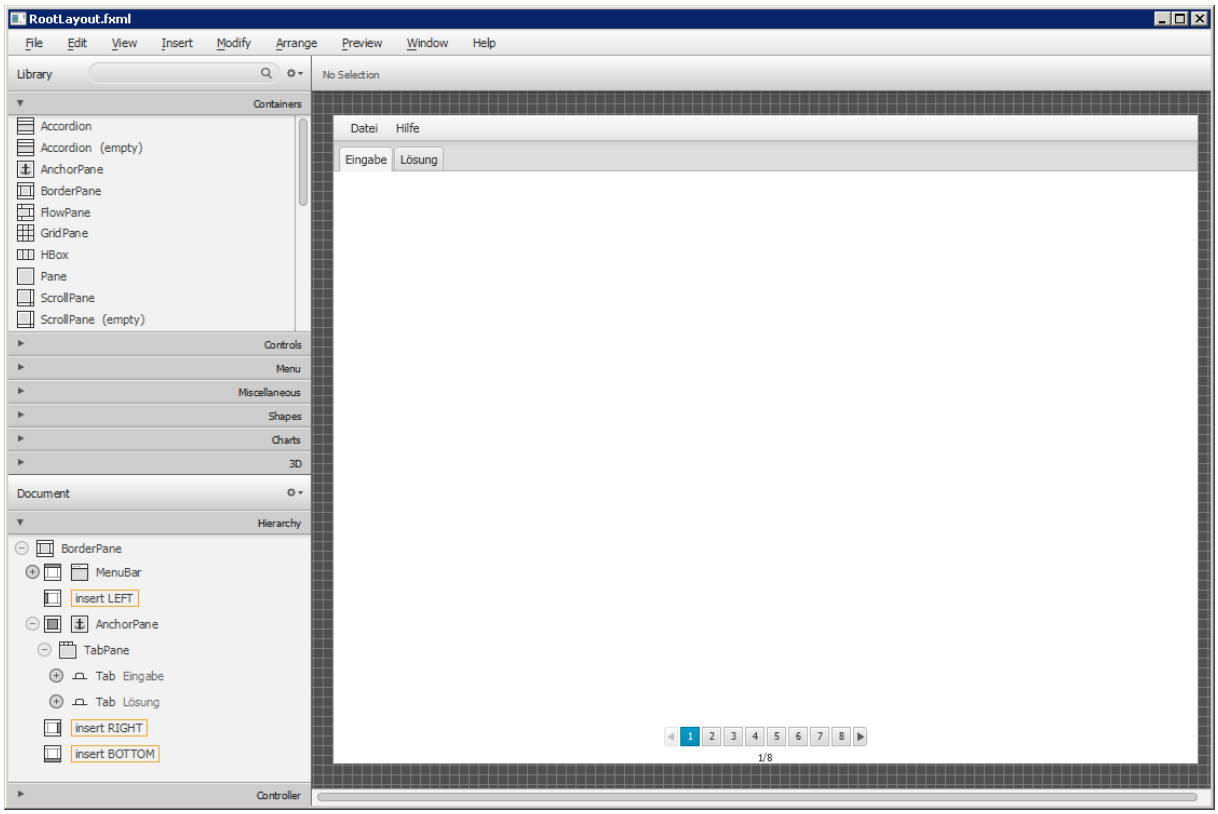

Abbildung 3: Der SceneBuilder

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 4: 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.

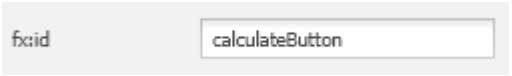

Abbildung 5: Setzten der fx:id im SceneBuilder

Und hier das Beispiel im Code:

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

Listing 1: Beispielprogramm

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

Apache Ant Apache Ant dient zur automatisierten Erstellung der ausführbaren Programme HPPLAN und CLSP aus einer gemeinsamen Code-Basis. Gleichzeitig werden Java-Docs erzeugt.

Die ausführbare Software wird mit Hilfe von Build-Skripte erstellt, die alle Informationen zu benötigten Bibliotheken, Quelldateien und sonstigen einzubindenen Dateien beinhalten. Nachfolgende Abbildung gibt eine Übersicht über die für Ant notwendigen Dateien.

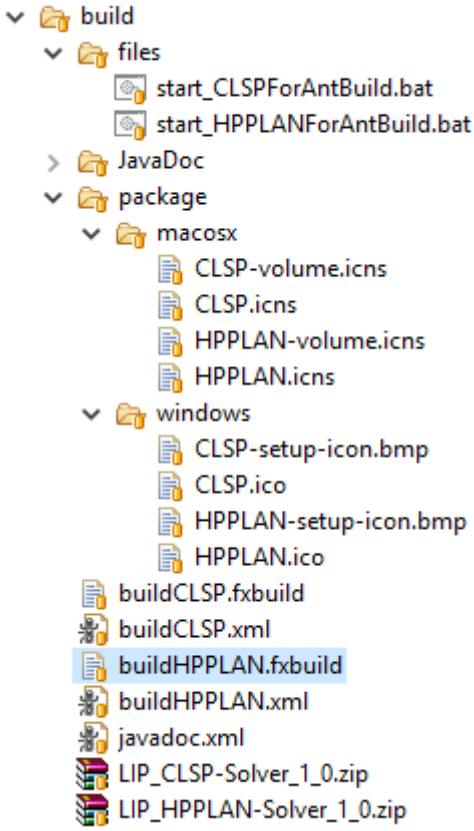

Abbildung 6: Dateien für Ant

Der Ordner “files“ enthält .bat Dateien für die Ausführung unter Windows. “JavaDoc“ enthält die erzeugte Dokumentation zu Klassen und Methoden. In “package“ sind Icon-Dateien für Windows und OSX hinterlegt. Mit Hilfe der Dateien “buildCLSP.xml“ und “buildHPPLAN.xml“ werden die Anwendungen erstellt. “javadoc.xml“ erzeugt die Java-Dokumentation.

Nachfolgender Auszug aus der Build-Datei zu HPPLAN zeigt,wie die Main-Klasse festgelegt wird.

```
1  <!-- XML ... -->
2
3  <fx:application id="fxApplication"
4      name="${toolVersioned}"
5      mainClass="de.oth.hsp.hpplan.HPPLANMain"
6      version="${version}"
7      toolkit="fx"
8  />
```

```
9
10    <!-- XML... -->
```

Listing 2: Ant Build-Skript HPPLAN

Es ist wichtig, dass im jeweiligen Build-Skript die JDK Version von Java 8 übergeben wird. Folgendes Listing zeigt den XML-Code im Skript.

```
1     <!-- XML ... -->
2
3     <javac includeantruntime="false" source="1.8" target="1.8"
4           srcdir="build/src" destdir="build/classes" encoding="UTF-8">
5       <classpath>
6         <fileset dir="build/libs">
7           <include name="*"/>
8         </fileset>
9       </classpath>
10    </javac>
11
12    <!-- XML ... -->
```

Listing 3: Ant Build-Skript mit Angabe der JDK Version

Ist dies gesetzt, können die Programme anschließend mit Hilfe der ausgegebenen BAT-Dateien gestartet werden.

2.2 Komponenten

2.2.1 Allgemein

Die unter der Komponente “common“ zusammengefasste “Komponenten“ enthalten die Hauptapplikation und geteilte Klassen für HPPLAN und CLSP.

common.dat Die Komponente “dat“ enthält die Komponenten “constraint“, “parser“, “parser.gen“ und “value“. Diese Komponenten sind notwendig für das Einlesen, Prüfen und Erstellen der DAT-Dateien, die dem ILOG-Framework übergeben werden.

common.ilog Festlegen von Schnittstellen für den Zugriff auf das Framework und die Entgegennahme von Ergebnissen von diesem. Weiterhin gibt es in der Komponente “common.ilog.exce
Klassen zur Fehlerbehandlung.

common.utils Diese Komponenten enthält Hilfsklassen für das Umwandeln von Arrays, dem Editieren einzelner Zellen in einer Tabelle und Dateioperationen.

common.view Enthält abstrakte Klassen und Methoden für Tabellen, die von den konkreten Klassen der View-Modelle von CLSP und HPPLAN geerbt werden, wenn diese eine Tabelle enthalten.

2.2.2 HPPLAN

Die Komponente “hpplan“ enthält für das HPPLAN-Tool notwendigen Komponenten.

hpplan.ilog Diese Komponente umfasst alle Klassen, die für die Berechnung mit dem ILOG-Framework notwendig sind. “HPPlanStatischModel“ beschreibt die MOD-Datei, die dem Framework übergeben wird. Weiterhin exisierten Klassen, die die Anfrage und das Rückgabeergebnis speichern. Die Klasse “HPPlanStatischSolvingAlgorithm“ ist zuständig für die Kommunikation mit ILOG. Die Klassen dieser Komponente implementieren dabei die Schnittstellen aus der Komponente “common.ilog“. Zusätzlich sind zwei Hilfsklassen für die Bildung von Produkten enthalten.

hpplan.model Enthält die Klasse “HpplanStatDatFile“, die eine Modell für das HPPLAN-Stat Problem enthält und vom ILOG-Framework zur Lösung des Problems benötigt wird.

hpplan.test Vorhanden, um die Anbindung an das Framework zu testen. Die enthaltene Klasse enthält Testdaten.

hpplan.view In dieser Komponente befinden sich die Controller und FXML Dateien der grafischen Oberfläche. Diese ist in ein RootLayout und zwei 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:

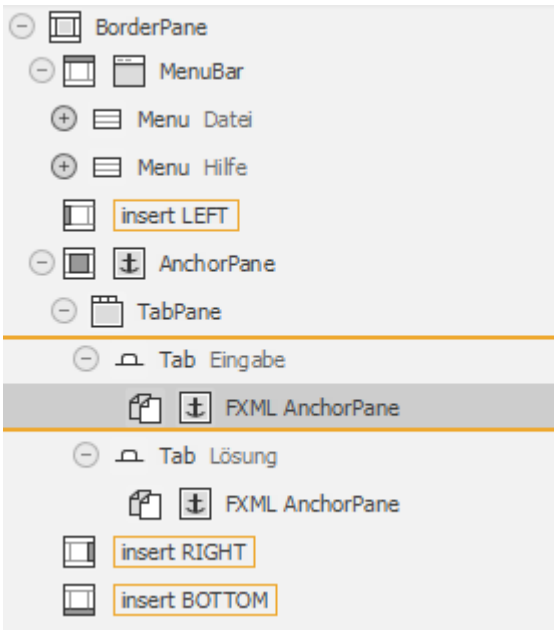

Abbildung 7: Darstellung der Reiter im SceneBuilder

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.

Zur Eingabe der Daten enthält der Tab “Eingabe“ 8 Unterseiten. Diese werden mit dem in Abbildung 1 am unteren Rand der Anwendung dargestellten PaginationController umgeschaltet. Dabei initialisiert und verwaltet die Klasse “PaginationController“ die Unterseiten. Mit Hilfe eines Listeners wird der Wechsel einer Seite erkannt und durchgeführt.

2.2.3 CLSP

Die Komponente “clsp“ enthält für das CLSP-Tool notwendigen Komponenten.

clsp.ilog Das CLSP-Model kann sowohl mit Gleitkommazahlen (Float) als auch Ganzzahlen (Integer) bestückt werden. Dafür gibt es zwei verschiedene Model-Klassen. Weiterhin exisitieren Klassen für die Anfrage und das Ergebnis des Algorithmus. Aufgrund der verschiedenen Modelle für CLSP gibt es auch zwei Klassen für die Kommunikation mit dem ILOG-Framework. Die genannten Klassen implementieren die dabei die allgemein definierten Schnittstellen aus “common.ilog“, wobei die Schnittstelle “ILogSolvingAlgorithm“ durch “ICLSPSolvingAlgorithm“ erweitert werden. Es gibt außerdem eine Hilfsklasse für die Produktbildung.

clsp.model Enthält die Klasse “ClspDatFile“, die eine Modell für das CLSP Problem enthält und vom ILOG-Framework zur Lösung des Problems benötigt wird.

clsp.test Vorhanden, um die Anbindung an das Framework zu testen. Die enthaltene Klasse enthält Testdaten.

clsp.view Siehe “hpplan.view“ unter Abschnitt 2.2.2.

Nachfolgende Abbildungen gibt einen Überblick über die im Package-Explorer zu sehenden Komponenten.

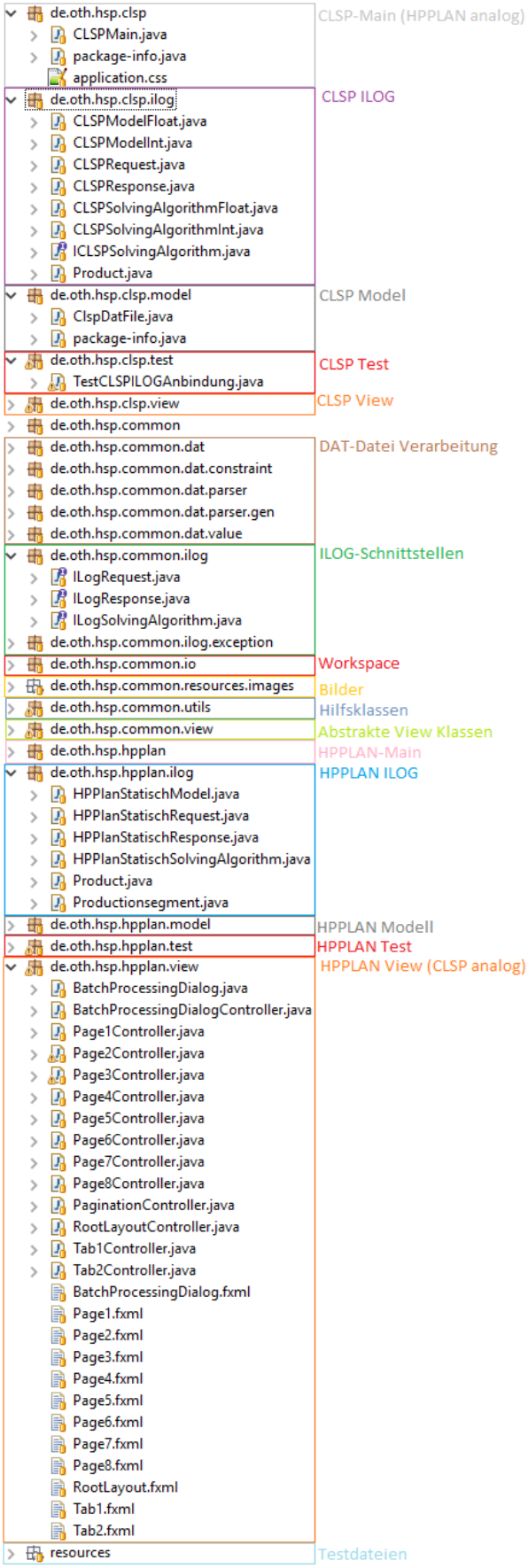

Abbildung 8: Komponenten im Package - Explorer

3 Implementierung

Dieses Kapitel beschreibt die Implementierung, sowie die bei der Entwicklung der Komponenten verwendeten Bibliotheken. 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:

JavaFX Zusätze: Die hier verwendeten Bibliotheken erweitern JavaFX um vorgefertigte Dialoge und Buttons. Weitere Informationen unter: http://fxexperience.com/controlsfx/

ILOG-Framework: Die ILOG-Framework Bilbiothek bildet die Schnittstelle zur ILOG Software.

ANTLR: ANother Tool for Language Recognition (ANTLR) ist ein leistungsfähiger Parser zum lesen, verarbeiten, ausführen oder übersetzen von strukturiertem Text oder Binärdateien.

Apache Ant: Die verwendeten Biliotheken werden für das Build-System Ant verwendet. Weitere Informationen unter: http://ant.apache.org/

Apache POI: Ist eine freie Java Library, welche den Umgang mit Exceldateien ermöglicht. Benötigt wird dies dann, wenn das Framework keine ILog Installation auf dem Rechner findet.

Nachfolgende Abbildung zeigt die verwendeten Bibliotheken in der Ordnerstruktur.

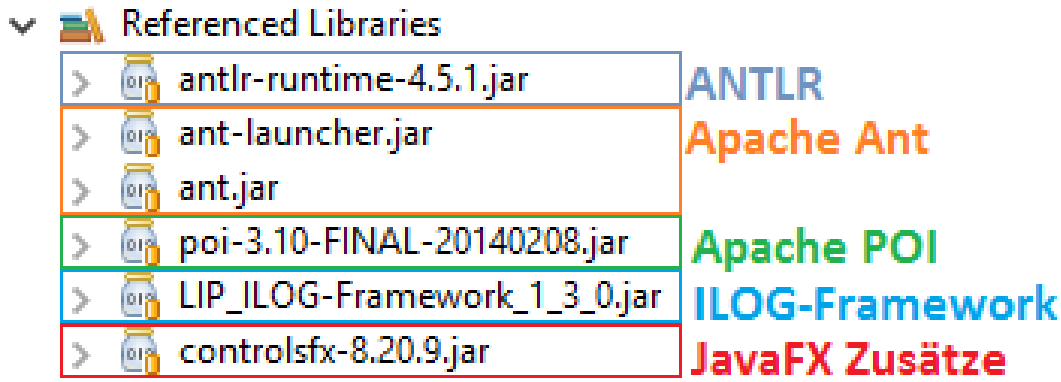

Abbildung 9: Verwendete Bibliotheken

In den folgenden Abschnitten werden einige ausgewählte Programmabläufe detaillierter beschrieben. Dabei steht vor allem der Umgang mit den DAT-Dateien und das Anstoßen der Berechnung im Vordergrund. Die Erzeugung von DAT-Dateien und das Laden dieser wird anhand von CLSP erklärt und ist zum Vorgehen in HPPLAN analog.

3.2 Erzeugung DAT-Dateien

Es besteht die Möglichkeit eingegebene Werte in einer DAT-Datei zu speichern und diese bei Bedarf wieder einzulesen (siehe 3.3). Das Speichern von Dateien wird dabei in der Klasse “RootLayoutController“ durchgeführt. Nachfolgend wird das Vorgehen der Erzeugung grob beschrieben.

1. Im Untermenü Datei wird die Funktion Speichern angewählt, wodurch im “RootLayoutController“ die Methode “onActionFileSave“ aufgerufen wird.
2. Es wird ein Dialog zum Speichern angezeigt, der dem Benutzer die Wahl über den Speicherort und den Dateinamen gibt.
3. Eine leere Datei vom Typ “.dat“ wird angelegt.
4. Alle im aktuellen CLSP-Modell enthaltenen Werte werden in der DAT-Datei gespeichert.

Die gespeicherten Dateien haben dabei nachfolgenden Inhalt und sind je nach eingegebenen Dateien befüllt.

```
1  CPLEX_EPGAP =0.0001;
2  // Anzahl Perioden:
3  T=2;
4  // Anzahl Produkte:
5  K=2;
6  //Ressource
7  J=2;
8  //grosse zahl
9  M=5768668;
10
11 //kapazitaeten
12 b= #[
13  1: [3000 3000]
14  2: [3000 3000]
15 ]#;
16
17
18 // Bedarfe je Produkt und Periode:
```

```
19 d= #[
20  1: [20  30]
21  2: [20  50]
22  ]#;
23  // Lagerkosten:
24  h= [2  2];
25  // Ruestkosten:
26  s= [200  200];
27  // Stueckbearbeitungszeit je Produkt und Ressource:
28  tb= #[
29  1: [20  30]
30  2: [20  50]
31  ]#;
32  // Ruestzeit je Produkt und Ressource:
33  tr= #[
34  1: [20  30]
35  2: [20  50]
36  ]#;
37  //mindestvorlaufzeiten
38  z= [0  0];
39  // Anfangslagerbestaende:
40  y0 = [10  0];
41  // Endlagerbestaende:
42  yT = [30  0];
```

Listing 4: Inhalt DAT-Datei

3.3 Laden von DAT-Dateien

Das Laden von Dateien wird ebenfalls mit durch die Klasse “” angetriggert. Im Zuge dessen wird die allgemeine Komponente “common.data“ und ihre Unterkomponenten zum Parsen der Dateien verwendet. Nachfolgend wird der Ablauf beschrieben.

1. Im Untermenü Datei wird die Funktion Öffnen angewählt, wodurch im “RootLayoutController“ die Methode “onActionFileOpen“ aufgerufen wird.
2. Es wird ein Öffnen-Dialog gestartet mit dem die zu öffnende Datei ausgewählt wird.
3. Mit Hilfe der in beiden Tools verwendeten Klasse “DatFileParser“ wird die gewählte Datei eingelesen und ein Modell vom Typ “ClspDatFile“ oder “HpplanStatDatFile“ zurückgegeben.
4. Die aktuelle Seite wird mit den neuen Werten befüllt. (Beim Aufruf einer Seite werden die Werte stets aus dem Modell geladen)

3.4 Berechnung

Nachfolgend werden sowohl für CLSP als auch HPPLAN grob die Berechnungschritte aufgezeigt. Die Berechnung wird in beiden Tools auf Seite 8 des Tabs Eingabe durch den Button “Berechnung starten“ ausgelöst. Dabei wird die Aktion an die Methode “calculateCLSP“ oder “calculateHPPLAN“ des “RootLayController“durchgereicht.

3.4.1 CLSP

Anders als bei HPPLAN kann im CLSP Tool eine weitere Einstellung vorgenommen werden. Unter Optionen\Algorithmuseinstellung kann zwischen Ganzzahl und Kommazahl in der Berechnung gewählt werden.

Nachfolgend werden die Schritte der Berechnung in der CLSP Anwendung aufgezeigt.

1. Das Modell wird erneut gespeichert, damit alle Werte aktuell sind.
2. Je nach Algorithmuseinstellung wird eine Instanz der Klasse “CLSPSolvingAlgorithmInt“ für die Ganzzahlberechnung oder “CLSPSolvingAlgorithmFloat“ für die Gleitkommaberechnung angelegt.
3. Das Modell wird in ein Request-Objekt “CLSPRequest“ geschrieben und der Algorithmus-Klasse zur Lösung übergeben.
4. Das Problem wird mit Hilfe von ILOG gelöst.
5. Die Lösung von ILOG wird in das Objekt “CLSPResponse“ geschrieben.
6. Das Ergebnis wird unter dem Tab “Lösung“ dargestellt.

3.4.2 HPPLAN

Nachfolgend werden die Schritte der Berechnung in der CLSP Anwendung aufgezeigt.

1. Das Modell wird erneut gespeichert, damit alle Werte aktuell sind.
2. Es wird ein Instanz der Klasse “HPPlanStatischSolvingAlgorithm“ erzeugt.
3. Das Modell wird in ein Request-Objekt “HPPlanStatischRequest“ geschrieben und der Algorithmus-Klasse zur Lösung übergeben.
4. Das Problem wird mit Hilfe von ILOG gelöst.
5. Die Lösung von ILOG wird in das Objekt “HPPlanStatischResponse“ geschrieben.
6. Das Ergebnis wird unter dem Tab “Lösung“ dargestellt.

3.5 Stapelverarbeitung

Der Menüpunkt Datei\Stapelverarbeitung ist in beiden Anwendungen vorhanden. Beim Auswählen des Menüpunktes wird ein Dialog geöffnet in dem mehrere Dateien zur Verarbeitung und ein Ziel zum Abspeichern der Ergebnisse ausgewählt werden können.
Es werden dabei bereits Ordner erzeugt und die Berechnung der Ergebnisse aus den gewählten DAT-Dateien ist implementiert. Jedoch ist es mit dem zum Zeitpunkt der Entwicklung zur Verfügung stehenden ILOG-Framework nicht möglich, die Ergebnisse in Excel-Dateien zu speichern. Dadurch ist es nicht möglich eine Stapelverarbeitung sinnvoll durchzuführen.

4 Abkürzungsverzeichnis

ANTLR ANother Tool for Language Recognition.

CLSP Capacitated Lot-Sizing Problem.

GUI Graphical User Interface.

HPPLAN Hauptproduktionsprogrammplanung.

XML Extensible Markup Language.