

„HTWG Konstanz – Fakultät Informatik

2015-01-21

Softwaredokumentation JobShopNeu

Dennis Klein, Wirtschaftsinformatik 7.Semester
Sebastian Stephan, Wirtschaftsinformatik 7.Semester

Inhaltsverzeichnis

Abbildungsverzeichnis	ii
1 Einführung und Ziele	1
2 Randbedingungen	3
2.1 Technische Einflussfaktoren.....	3
2.2 Organisatorische Einflussfaktoren.....	3
3 Projekt-Ist-Zustand bei Übernahme (22.10.2014)	4
4 Probleme	5
4.1 Lösungsansätze	5
5 Kontextabgrenzung	6
6 Softwarearchitektur	7
6.1 Klassendiagramm: JobShopApp.....	7
6.2 Klassendiagramm: JobShopView	8
6.3 Klassendiagramm: JobShopCalculator	9
6.4 Klassendiagramm: JobShopNumberFormatter.....	10
6.5 Klassendiagramm: JobShopFileFilter.....	10
6.6 Klassendiagramm: JobShopAboutBox.Java.....	11
6.7 Klassendiagramm: JobShopResultBox.....	12
6.8 Klassendiagramm: ResultList.....	13
6.9 Klassendiagramm: Result	13
7 Softwarequalität	14
8 Begründung von Entwurfsentscheidungen	15
8.1 Problemerkörterung	16
8.2 Lösungsansatz.....	16
8.3 Vorgehensweise zur Erstellung der 64 Bit-Applikation:.....	17
9 Weiterentwicklung JobShopNeu	19
10 Beispielaufgabe	23
10.1 Lösung traditionell mit lpSolve:	24

10.2	Lösung mit JobShopNeu:.....	24
------	-----------------------------	----

Abbildungsverzeichnis

Abbildung 1: Klassendiagramm (22.10.2014)	4
Abbildung 2: Kontextdiagramm.....	6
Abbildung 3: Klassendiagramm JobShopApp	7
Abbildung 4: Klassendiagramm JobShopView	8
Abbildung 5: Klassendiagramm JobShopCalculator	9
Abbildung 6: Klassendiagramm JobShopNumberFormatter	10
Abbildung 7: Klassendiagramm JobShopFileFilter	10
Abbildung 8: Klassendiagramm JobShopAboutBox	11
Abbildung 9: Klassendiagramm JobShopResultBox	12
Abbildung 10: Ergebnisausgabe.....	12
Abbildung 11: Klassendiagramm ResultList	13
Abbildung 12: Klassendiagramm Result.....	13
Abbildung 13: DLL in Projekt importieren.....	17
Abbildung 14: Runnable JAR File Export	18
Abbildung 15: Projekthinhalt	18
Abbildung 16: Aufbau Start_Job_Shop_64Bit.bat-Datei.....	19
Abbildung 17: Programmaufruf	19
Abbildung 18: Eingabematrix 16/16	20
Abbildung 19: Methode initComponents()	20
Abbildung 20: Popup Beispiel bei Falscheingabe.....	21
Abbildung 21: Methode stringValue(String s)	21
Abbildung 22: Eingabematrix JobShopNeu.....	22
Abbildung 23: Lösungsmatrix.....	22
Abbildung 24: FileWriter-Erweiterung.....	22
Abbildung 25: Lösungsvektor lpSolve.....	23
Abbildung 26: Eingabematrix JobShopNeu.....	24
Abbildung 27: Lösungsmatrix JobShopNeu	24

1 Einführung und Ziele

Im Rahmen der betrieblichen Systemforschung pflegt die Hochschule für Technik, Wirtschaft und Gestaltung (HTWG) seit vielen Jahren eine Methodenbank. In der Methodenbank sind verschiedene Programme für die Lösung unterschiedlicher Problemstellungen der linearen Programmierung (LP) enthalten. Die verschiedenen Software-Anwendungen greifen dabei auf unterschiedliche Solver zur Lösung der erstellten LP-Modelle zurück. Die Methodenbank umfasst sowohl Open-Source-Solver (lpSolve, MOPS) als auch kommerzielle Solver-Produkte (IBM ILOG CPLEX).

Entwickelt und auch eingesetzt wird diese Methodenbank primär in der Vorlesung „Operations Research“. Das Ziel der Weiterentwicklung und Sicherstellung der Lauf- und Funktionsfähigkeit der einzelnen Methoden und Programmen und der Methodenbank selbst ist Gegenstand der Veranstaltungen „Teamprojekt“ und „ALO“ in den höheren Wirtschaftsinformatik-Semestern.

Bei JobShop handelt es sich um ein Programm dessen Aufgabe darin besteht, eine vorgegebene Zahl von Produkten hinsichtlich einer optimalen, kürzesten Gesamtdurchlaufzeit auf verschiedene Maschinen einzuplanen. Derzeit befindet sich sowohl JobShop als auch JobShopNeu¹ weder in der Methodenbank OR_ALPHA noch in ORWEB da die Programme trotz vollständiger Installation unter Windows 7 64-Bit nicht gestartet werden können. Gemäß dem Vorgängerprojekt lässt sich JobShopNeu jedoch ab und zu unter Windows 7 32-Bit starten, was für die Hochschule jedoch nicht ausreichend ist.

Primärziel des Projekts ist es, JobShopNeu auf Windows 7 SP1 (32-/64-Bit-Architektur) lauffähig zu bekommen sowie ggf. die „umfangreiche“ Installation des Programms zu verbessern. Ist die Software lauffähig, soll das System dem Benutzer die Möglichkeit bieten, neue Maschinen und Produkte anzulegen und die entsprechenden Durchlaufzeiten zu hinterlegen. Es soll die Möglichkeit bestehen, diese sowie gespeicherte LP-Modelle abzuspeichern bzw. zu laden (z.B. XML- oder TXT-Datei). Über eine Schnittstelle zu einem Solver, primär dem lpSolve, soll ein generiertes LP-Modell gelöst werden. Die Einbindung weiterer Solver ist durchaus denkbar (z.B. IBM ILOG CPLEX). Das Ergebnis wird dann dem Benutzer in einem neuen Fenster textuell oder graphisch angezeigt. Durch die Angabe eines Dateiverzeichnisses soll das Ergebnis als XML-, TXT-, CSV- oder ähnlichem Dateiformat dauerhaft gesichert werden können. Außerdem wird eine Eingabeüberprüfung insb. bei der

¹ Bei JobShopNeu handelt es sich um die Neuimplementierung von JobShop in Java.

Eingabe der Durchlaufzeiten angestrebt. Dadurch sollen Falscheingaben vorgebeugt und die Software benutzerfreundlicher gestaltet werden.

Parallel soll versucht werden, JobShop, die ältere Version von JobShopNeu, auf Windows 7 SP1 (32-/64-Bit-Architektur) oder über eine virtuelle Maschine wie z.B. DOSBox lauffähig zu bekommen, sodass den Studierenden übergangsweise ein Softwaretool für Optimierungsprobleme dieser Art zur Verfügung steht. Eine Weiterentwicklung dieses Tools ist nicht Bestandteil dieses Projekts.

Welche Anforderung in welchem Umfang und Detail erbracht werden sollen, wurde zusammen mit Prof. Dr. Grütz und dem Projektteam mündlich besprochen und in einem Pflichtenheft festgehalten. Für die Bewertung des Projekterfolgs sowie die Benotung soll zusätzlich auf dieses Dokument verwiesen werden. Die in diesem Dokument erwähnten Funktionalitäten dienen der langfristigen Orientierung hinsichtlich der Weiterentwicklung von JobShopNeu bis hin zur Einbindung in OR_ALPHA und ggf. ORWEB.

Ziel dieser Softwaredokumentation ist es, den aktuellen IST-Zustand des JobShopNeu zu erfassen. Da keine Softwaredokumentation von JobShop als auch JobShopNeu vorhanden war, soll der aktuelle IST-Zustand des Systems, das Zusammenspiel der einzelnen Klassen (und Komponenten) sowie die aktuellen Probleme analysiert und festgehalten werden. Auf dieser Basis sollen dann die Weiterentwicklungsmaßnahmen (Entwurfsentscheidungen) getroffen werden.

2 Randbedingungen

Im Folgenden werden sämtliche (externe) Faktoren genannt, welche die (neue) Architektur beeinflussen oder einschränken können. Auf Basis dieser Faktoren sollen schließlich unter anderem alle anwendungs- und problembezogenen Entwurfsentscheidungen begründet werden.

2.1 Technische Einflussfaktoren

Betriebssystem: Windows 7 SP1 (32-/64-Bit-Architektur)

Programmiersprache: Java

Entwicklungsumgebung: Eclipse Juno JDK1.8

Komponenten/Bibliotheken:

1. Sämtliche Java AWT-/Swing-Komponenten:
 - 1.1. AbsoluteLayout.jar
 - 1.2. appframework-1.0.3.jar
 - 1.3. swing-worker-1.1.jar
2. Schnittstelle für Solver
 - 2.1. lpsolve55j.jar

2.2 Organisatorische Einflussfaktoren

Vorgehensmodell: Iteratives und inkrementelles Vorgehen, wöchentliche Teambesprechungen sowie Projektbesprechungen zusammen mit Prof. Dr. Grütz

Projektzeitraum: 22.10.2014 – 21.01.2015

Personal: 2 Java-Entwickler

Qualitätsstandards: Sollen der Messung der Softwarequalität dienen:

1. ISO 9126 bzw. ISO 25010 und
2. Oracle-Code-Convention

3 Projekt-Ist-Zustand bei Übernahme (22.10.2014)

Das JobShopNeu-Javaprogramm wurde in der Entwicklungsumgebung „Netbeans IDE 6.8“ programmiert. Als Solver wurde lp_solve 5.5 verwendet. Entwickelt und getestet wurde die Software unter Windows, wobei die Lauffähigkeit nicht dauerhaft gewährleistet werden konnte.

Momentan besteht die Software aus 10 Klassen.²

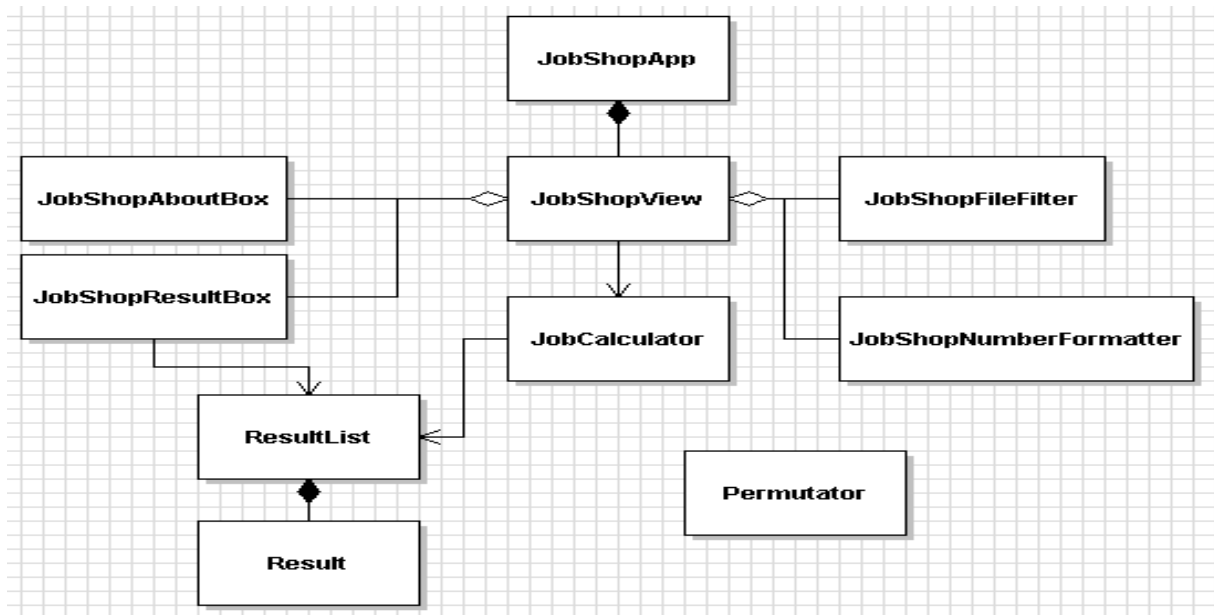


Abbildung 1: Klassendiagramm (22.10.2014)

Laut der „Dokumentation“ des Vorgängerprojekts lässt sich die Software potentiell auf einer 32-Bit-Architektur starten sodass sich der Programmablauf wie folgt beschreiben lassen kann:³

Im Hauptfenster des JobShops (JobShopView) können die Bearbeitungsdauern der Produkte für jede Maschine eingetragen werden. Dabei kontrolliert der JobShopNumberFormatter, dass es sich bei den Einträgen um Zahlen zwischen 0 und 99 handelt.

Beim Klick auf den Berechnen-Button wird der JobCalculator aufgerufen. Der JobCalculator erstellt dann das LP-Modell und berechnet ein optimales Ergebnis mit dem lpSolve. Dieses Ergebnis speichert er dann in der ResultList ab, in welcher alle Punkte auf welcher Maschine welches Produkt an welchem Zeitpunkt gebaut wird (Result), gespeichert werden. Die ResultList wird dann an die JobShopResultbox übergeben, welche dann das Ergebnis anzeigt.

² vgl. Dokumentation – JobShop2.docx (WS 2010/11)

³ vgl. Dokumentation – JobShop2.docx (WS 2010/11)

Der JobShopFileFilter kontrolliert, dass nur (Text-)Dateien gespeichert/geladen werden, deren Bezeichnung mit „job“ endet.

Die JobShopAboutBox zeigt ein Fenster an in welchem Informationen wie Autor und Erstellungsdatum angegeben sind.

Die Permutator-Klasse ist ein erster Entwurf um das Job-Shop-Problem so zu lösen, wie in der alten JobShop-Version.

Aus diesem Grund werden zunächst die Klassen sowie ihre Attribute näher beschrieben und deren Beziehung zueinander.

4 Probleme

Da dem Projektteam während des Projektzeitraums kein Rechner bzw. eine Virtual Machine (VM) mit Windows 7 SP1 32-Bit-Architektur zur Verfügung stand, konnte sich der soeben geschilderte Programmablauf zunächst nicht validieren lassen. Erst im späteren Verlauf konnte eine passende VM organisiert werden. Auf dieser wurde versucht den geschilderten Programmablauf zu validieren, jedoch vergebens.

Abbildung 2 skizziert grob wie die einzelnen Klassen miteinander in Beziehung stehen. Jedoch über die Funktionsweise sowie der Softwarequalität sagt dieses Klassendiagramm sowie die mitgelieferte Dokumentation nur wenig aus.

Die daraus resultierenden Folgen für das Projektteam waren

- eine aufwändige Codeanalyse,
- Bibliotheksabhängigkeiten aufgrund der zuvor verwendeten Entwicklungsplattform Netbeans IDE 6.8 sowie
- die Identifizierung möglicher Ansatzpunkte für Verbesserungen bzw. Erweiterungen.

4.1 Lösungsansätze

Es wurden zu Beginn des Projekts drei mögliche Ansätze hinsichtlich der Ausführbarkeit des Programms diskutiert:

1. Dynamische Nutzung eines bzw. mehrerer externer Solver (IpSolve, IBM ILOG CPLEX, MOPS, ...) über entsprechende Schnittstellen⁴

⁴ Wobei primär der Fokus auf dem Open Source Produkt IpSolve lag (auch in Hinblick auf die Veröffentlichung in ORWEB).

- Der Programmcode nutzte bereits die Schnittstelle des lpSolve, jedoch war eine Ausführung auf Grund einiger Exceptions hinsichtlich des lpSolve nicht möglich. Daraufhin wurde eine aufwändige Fehleranalyse betrieben.
2. Statische Einbindung des lpSolve in das Softwareprojekt
 - Eine weitere Möglichkeit war die statische Einbindung des lpSolve gewesen, dadurch müsste dieser jedoch fest ins Projekt eingebunden werden, ein fester Pfad gesetzt werden und der Code umgeschrieben werden.
 3. Einbindung von Google's linearen Programmierungssystem "GLOP"
 - Der Google GLOP Solver kam insofern in Frage, dass dieser numerisch sehr stabil und schnell in der Berechnung ist und ressourcensparend arbeitet. Die Einbindung erfordert jedoch einen hohen Mehraufwand, da der Code komplett auf den neuen Solver umgeschrieben bzw. ergänzt werden müsste für die Erstellung eines Solver konformes LP-Modells.

5 Kontextabgrenzung

Beschreibung des Systems aus Black-Box-Sicht. Sämtliche Interaktion von außen mit dem System sowie einzelner System-Komponenten soll übersichtlich dargestellt werden.

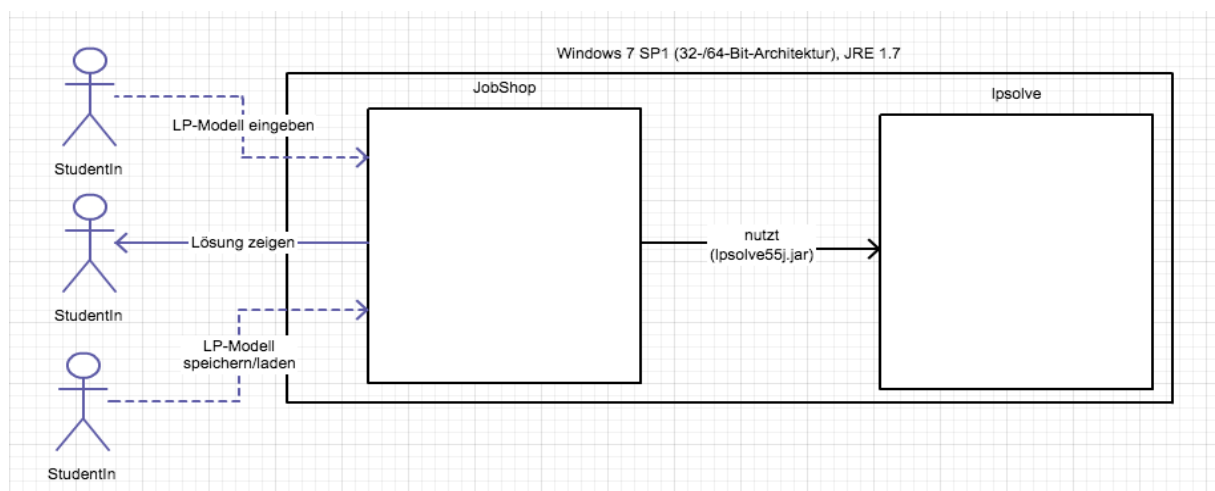


Abbildung 2: Kontextdiagramm

6 Softwarearchitektur

Im folgendem wird die Ist-Architektur in Form von Klassendiagrammen sowie einer kurzen Beschreibung dieser festgehalten. Der Hintergrund dieses Projektschritts war wie bereits erwähnt die unzureichende Softwaredokumentation des Entwicklerteams.

6.1 Klassendiagramm: JobShopApp

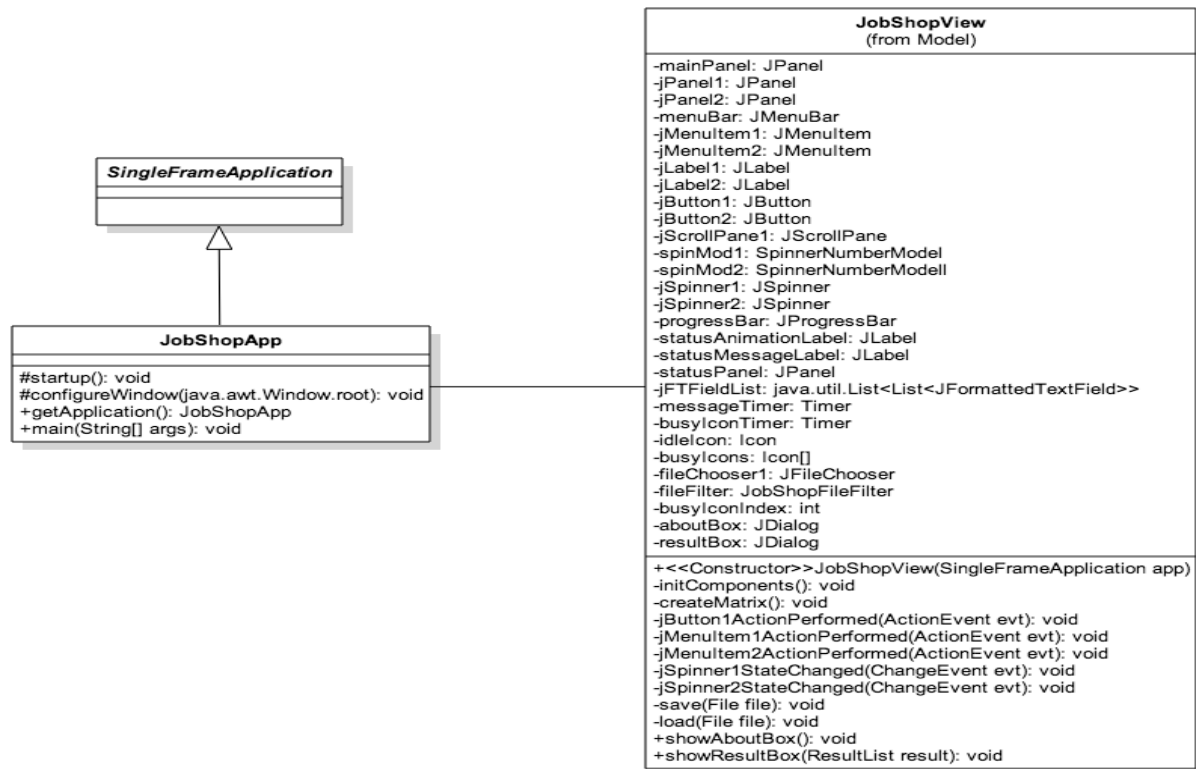


Abbildung 3: Klassendiagramm JobShopApp

Diese Klasse dient der Instanziierung und Verwaltung der GUI. Durch die Verwendung des Singleton-Patterns soll sichergestellt werden, dass von einer Klasse nur ein Objekt (Instanz) existiert. Dadurch soll eine bessere Zugriffskontrolle bzw. eine bessere Wartbarkeit der einzelnen Objekte gewährleistet werden. **JobShopApp** erbt dabei von der abstrakten Klasse *SingleFrameApplication*, die Methoden zur Verwaltung (WindowListener, Daten laden/speichern, Anwendung öffnen/schließen usw.) der instanziierten Session bereitstellt. Über die Methode *launch()* wird die Klasse instanziiert. Diese ruft die Methode *getApplication()* auf, die eine bestehende Instanz zurückgibt bzw. falls noch keine existiert, eine Application-Instanz zurückliefert. Über *startup()* wird die GUI erzeugt und angezeigt. Die Erstellung und Abbildung der Eingabelogik für das LP-Modell wurde in eine weitere Klasse ausgelagert, der **JobShopView**-Klasse.

6.2 Klassendiagramm: JobShopView

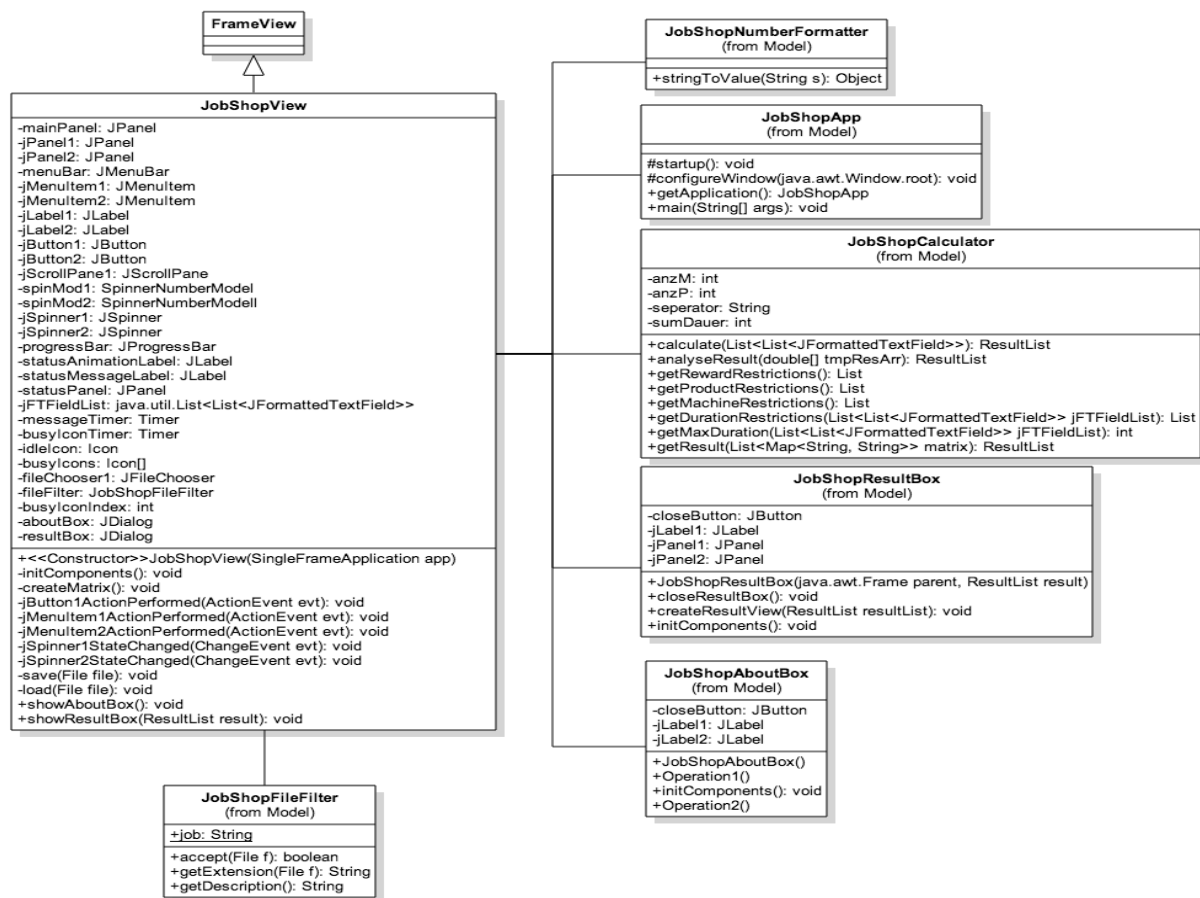


Abbildung 4: Klassendiagramm JobShopView

In dieser Klasse sind sämtliche Attribute und Methoden für die Erzeugung der GUI implementiert. Die Methode `initComponents()` kümmert sich um die Initialisierung der entsprechenden GUI-Attribute wie der Menüleiste (`JMenu()`) oder der Eingabematrix (`createMatrix()`), die dem Benutzer angezeigt werden soll. In einer ResourceMap werden die Namen und Bezeichnungen der jeweiligen Swing-Komponenten verwaltet und gesetzt. Diese ResourceMap befindet sich in einem separaten Ordner mit den entsprechenden Properties-Dateien. Bei einer Java Properties-Datei handelt es sich um eine einfache Textdatei, die in Java als einfache Konfigurationsdatei verwendet wird. Des Weiteren werden noch passende EventListener implementiert um zum Beispiel die Maschinen- bzw. Produktanzahl der Eingabematrix zur Laufzeit anzupassen. Diese Triggern dann die entsprechenden Funktionen wie z.B. speichern und laden der Eingabematrix, Erstellung des LP-Modells und dessen Berechnung.

6.3 Klassendiagramm: JobShopCalculator

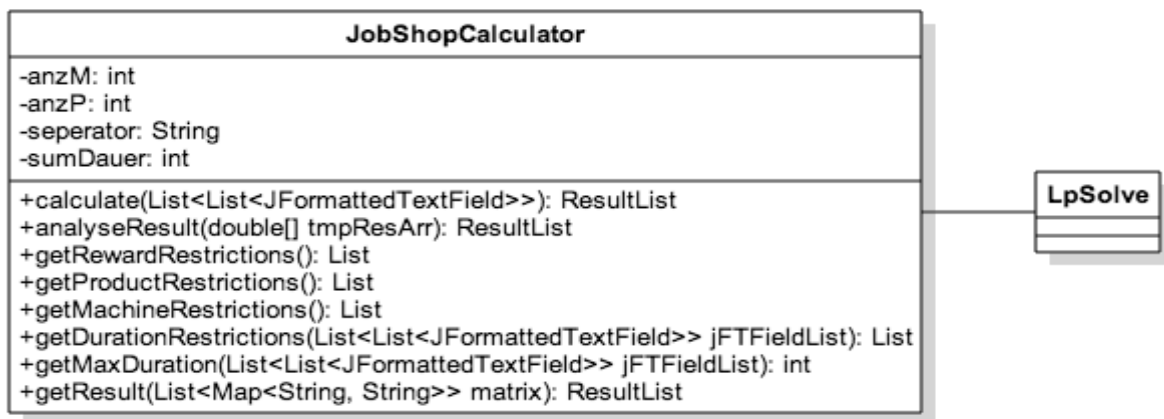


Abbildung 5: Klassendiagramm JobShopCalculator

Die Klasse JobShopCalculator überführt nun die Benutzereingabe in ein LP-Modell. Dabei werden folgende Restriktionen durch entsprechende Methodenaufrufe erstellt:

- *getDurationRestriction(List<List<JFormattedTextField>> jFTFieldList)*: Jedes Produkt muss innerhalb der Gesamtdurchlaufzeit abgearbeitet werden. Jedes Produkt hat so viel Variablen wie die Gesamtdurchlaufzeit beträgt. Natürlich wird aufgrund zeitlicher Grenzen des Produkts nicht der gesamte Variablenbereich benutzt.
- *getMachineRestriction()*:
Jede Maschine kann pro Zeiteinheit immer nur 0 oder 1 Produkt bearbeiten.
- *getMaxDuration()*:
Ermittelt die Gesamtdurchlaufzeit, in welcher die Produkte abgearbeitet werden müssen und gibt diese als Summe zurück. Diese ist notwendig um die Gesamtanzahl der benötigten Strukturvariablen als auch der benötigten Restriktionen zu ermitteln.
- *getProductRestriction()*:
Ein Produkt kann pro Zeiteinheit immer nur von 1 Maschine bearbeitet werden.
- *getRewardRestriction()*:
Die Produkte sollen möglichst an einem Stück bearbeitet werden. Dadurch soll ein ständiger Wechsel auf den Maschinen in Hinblick auf potenziell anfallende Rüstkosten vermieden werden.

Anschließend wird das zu lösende LP-Modell aus den einzelnen Restriktionslisten durch *calculate(List<JFormattedTextField>> jFTFieldList)* zusammengebaut, an den Solver übergeben, gelöst und das Ergebnis als Result zurückgegeben.

6.4 Klassendiagramm: JobShopNumberFormatter

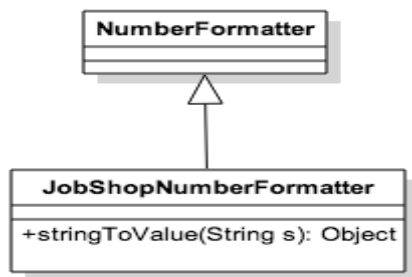


Abbildung 6: Klassendiagramm JobShopNumberFormatter

In dieser Klasse werden die Eingaben des Benutzers in der Eingabematrix entsprechend den Anforderungen, dass die Durchlaufzeiten nur ganzzahlig sein dürfen sowie die Durchlaufzeit nicht 99 Zeiteinheiten überschreiten darf, validiert.

6.5 Klassendiagramm: JobShopFileFilter

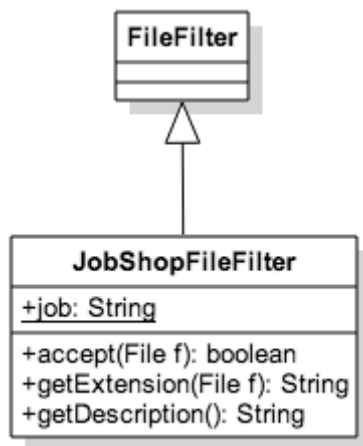


Abbildung 7: Klassendiagramm JobShopFileFilter

Klasse, die für das Erstellen und Verarbeiten von job-Dateien zuständig ist. Ermöglicht das Speichern der Eingabematrix automatisch als job-Datei. Im Prinzip handelt es sich dabei um eine einfache txt-Datei. Des Weiteren wird beim Laden überprüft, dass nur job-Dateien in die Anwendung geladen werden können.

6.6 Klassendiagramm: JobShopAboutBox

Die Klasse JobShopAboutBox zeigt grundlegende Metainformationen über die JobShopNeu Applikation:

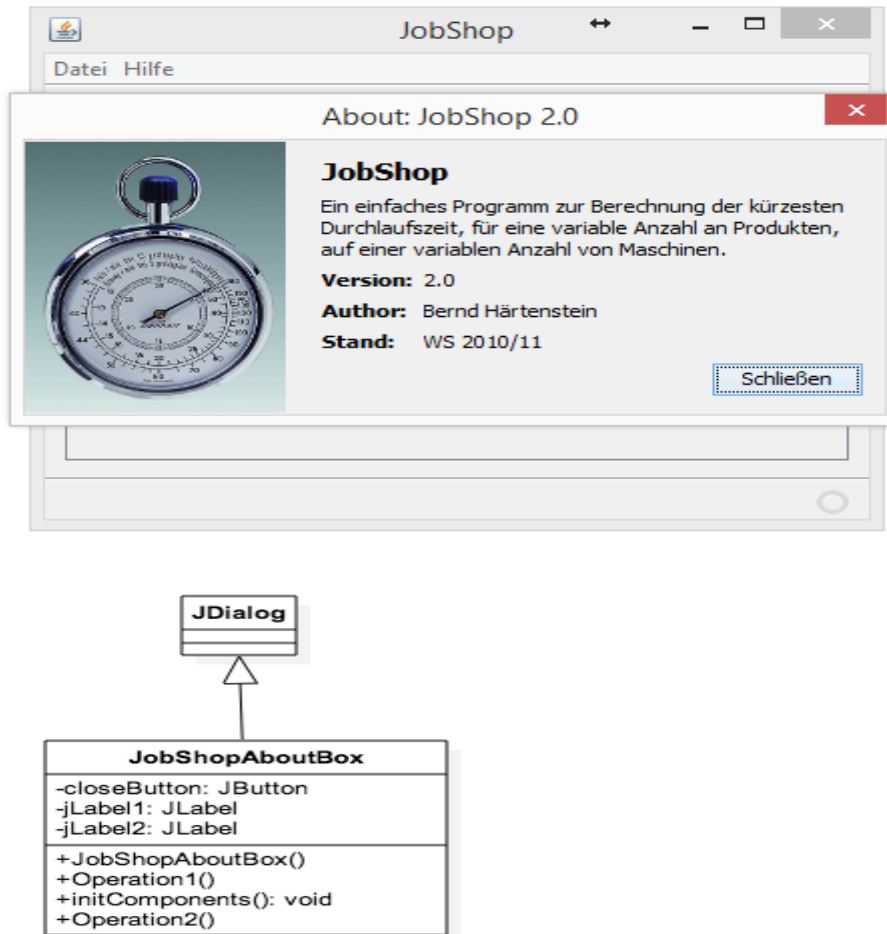


Abbildung 8: Klassendiagramm JobShopAboutBox

Die Klasse JobShopView ruft die Methode `showAboutBox()` auf, woraufhin sofern noch nicht vorhanden, ein Frame erstellt wird und der Klasse JobShopAboutBox als Parameter übergeben wird. Andernfalls wird die AboutBox direkt angezeigt.

```
public void showAboutBox() {  
    if (aboutBox == null) {  
        JFrame mainFrame = JobShopApp.getApplication().getMainFrame();  
        aboutBox = new JobShopAboutBox(mainFrame);  
        aboutBox.setLocationRelativeTo(mainFrame);  
    }  
    JobShopApp.getApplication().show(aboutBox);  
}
```

Die Klasse JobShopAboutBox selbst erbt von der Klasse `javax.swing.JDialog`. Der `JDialog` ähnelt sehr stark einem `JFrame` mit der Ausnahme, dass der `JDialog` modal gesetzt werden

kann. Modal bedeutet, dass, während der betreffende *JDialog* angezeigt wird, kein anderes Fenster verwendbar bzw. aktivierbar ist. Über die Methode *initComponents()* werden verschiedene JLabel sowie ein JButton initialisiert. Der JButton dient zum Schließen des Fensters, die JLabel für die Angaben zur Beschreibung, der Version, des Autors, sowie dem Stand, was in obiger Abbildung sehr gut zu erkennen ist. Mit Hilfe des *dispose* Befehls, werden die zugeordneten Windows-Ressourcen wieder freigegeben.

6.7 Klassendiagramm: JobShopResultBox

Nachdem das gefundene Optimum berechnet wurde, muss dieses anschließend dargestellt werden. Die Klasse *JobShopResultBox* erbt daher von der Klasse *javax.swing.JDialog* und initialisiert alle nötigen Komponenten um das Ergebnis, wie in nachfolgender Abbildung darzustellen:

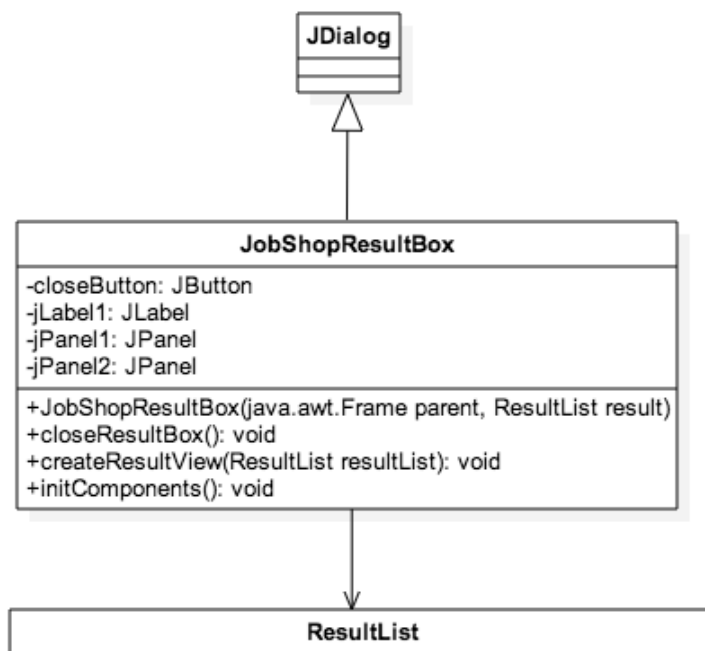


Abbildung 9: Klassendiagramm *JobShopResultBox*



Abbildung 10: Ergebnisausgabe

6.8 Klassendiagramm: ResultList

Die Klasse ResultList deklariert eine List vom Typ Result sowie die Variablen anzM (Anzahl Maschinen), anzP (Anzahl Produkte) und maxDauer (maximale Dauer), welche zur Sammlung gefundener Ergebnisse nötig sind.

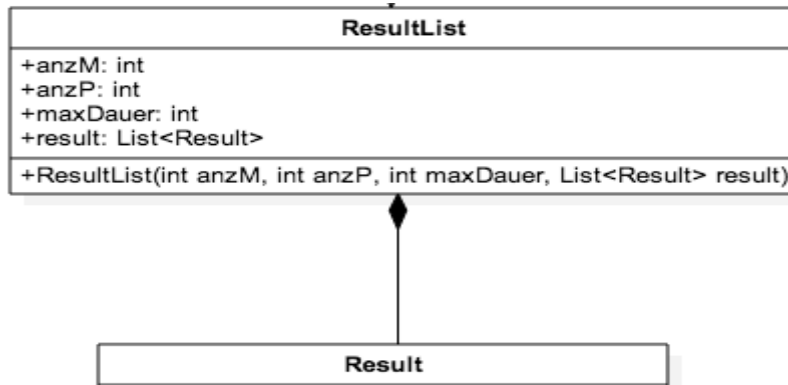


Abbildung 11: Klassendiagramm ResultList

6.9 Klassendiagramm: Result

Um ein gefundenes Ergebnis darstellen zu können, werden in der Klasse Result die Integer Variablen zeitpunkt, maschine und produkt deklariert.

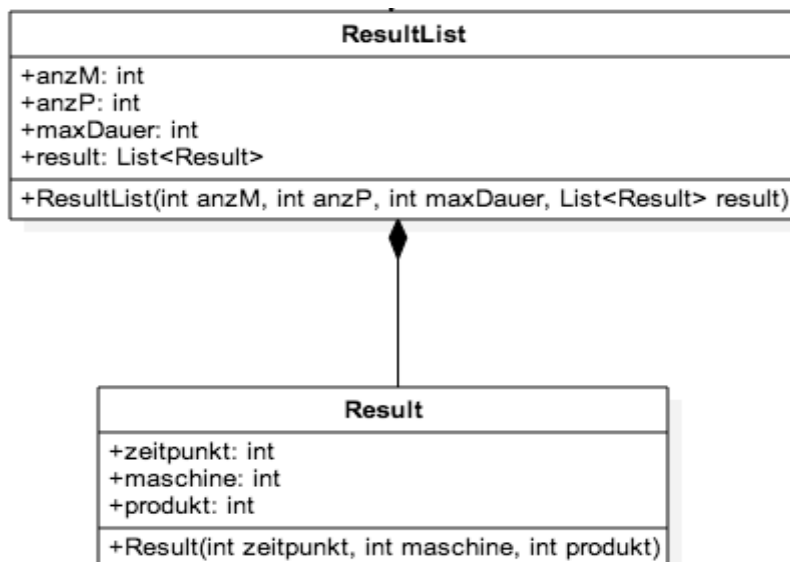


Abbildung 12: Klassendiagramm Result

7 Softwarequalität

Da keine Systemspezifikation vorhanden war, wurde anhand der internationalen Norm für System und Software-Engineering – Qualitätskriterien und Bewertung von System und Softwareprodukten (ISO 25010) versucht die Software hinsichtlich folgender Qualitätskriterien zu beurteilen:

- Functional suitability

Zu Beginn konnte sich nur wenig über die funktionale Vollständigkeit sagen, da sich die Software nicht hat ausführen lassen. Jedoch zeichnete sich durch den Modellierungsprozess der einzelnen Klassen ab, dass ein fachlich korrektes LP-Modell gemäß den Anforderungen, die im Commitment spezifiziert wurden, erstellt wird (Konformität). Die Erstellung, Berechnung und Ergebnisanzeigen wurden in entsprechende Methoden ausgelagert. Durch die Verwendung der von lpSolve spezifizierten Klassenbibliothek lpsolve55.jar wurde die Anwendung vom Solver entkoppelt, was die Interoperabilität erhöhte. Ein spezifisches Exception-Handling wurde in Form von try-catch-Statements konsequent durchgeführt sodass eine entsprechende Programmsicherheit gewährleistet werden konnte.

- Reliability

Es findet eine Überprüfung der Benutzereingaben statt. So lassen sich beliebig viele Maschinen und Produkte anlegen, jedoch nur bis zu einem Maximum von jeweils 16 Maschinen und Produkten. Außerdem lassen sich nur ganzzahlige Durchlaufzeiten im Intervall von [0,99] angeben. Jedoch steigt der Ressourcenverbrauch und somit die Berechnungsgeschwindigkeit mit steigender Größe des LP-Modells. Das liegt zum einen an den vielen Anforderungen in Form von Restriktionen⁵ aber auch am lpSolve. Da es sich um ein Open Source Produkt handelt ist dieser nun mal nicht so leistungstark wie z.B. ein kommerzieller IBM ILOG CPLEX.

- Usability

Die GUI ist einfach gehalten und bietet alle Grundfunktionalitäten an, so dass die Software schnell erlernbar ist trotz keiner Benutzeranleitung. Durch eine implementierte Eingabevalidierung werden entsprechende Nutzungshinweise dem Benutzer angezeigt.

- Portability

Da es sich um eine Neuimplementierung in Java handelt lässt sich die Software

⁵ So besitzt z.B. eine 2x3 Matrix 132 Variablen + 127 Restriktionen

prinzipiell auf jedem Betriebssystem ausführen. Da jedoch auf ein externer Solver zur Lösung der LP-Modelle zurückgegriffen wird, muss neben der entsprechenden JRE auch eine vom Betriebssystem unterstützte Version des lpSolve installiert sein, was die Portabilität des Softwaretools in der Hinsicht einschränkt, dass zusätzlich der lpSolve installiert sein muss. So muss z.B. unter Windows die entsprechende DLL-Datei (32-/64-Bit-Architektur) im Projekt eingebunden sein. Dadurch entstanden zwei Softwareprodukte, ein JobShopNeu für Win7 32-Bit-Architektur und ein JobShopNeu für Win7 64-Bit-Architektur.

- Maintainability

NetBeans nutzt keine proprietären Klassen, sodass der Programmcode 1:1 auch in ein Eclipse-Projekt kopiert werden kann und dort ohne Anpassung läuft. Jedoch mussten einzelne Klassenbibliotheken nachgeladen werden da diese ansonsten nicht exportiert werden konnten.

8 Begründung von Entwurfsentscheidungen

Wie bereits in Kapitel 4.1 beschrieben, wurde primär der erste Lösungsansatz verfolgt. Hintergründe für diese Entscheidung waren vor allem

- Wiederverwendbarkeit des bestehenden Softwarecodes
- Entkopplung zwischen Java-Anwendung und konkreten Solver
- Wartbarkeit

Gleichwohl wurden auch Ansätze in Bezug auf die Lösungsansätze 2 und 3 ausgearbeitet, da selbst nach 2 Monaten sich kaum Projektfortschritte in Hinblick auf Lösungsansatz 1 ergaben. Für Lösungsansatz 2 wurde ähnlich wie im Softwareprodukt „Diätplaner“ versucht, den lpSolve direkt in das Projekt einzubinden. Dabei müsste wie bereits in unserem Java-Projekt das LP-Modell z.B. in Form einer Matrix dargestellt werden. Diese müsste zusätzlich noch um die Zielfunktion (MIN/MAX!) ergänzt werden. Anschließend müsste in einer neuen Klasse (z.B. LpSolve.java) der in das Projekt eingebundene lpSolve statisch über eine (oder mehrere) Pathvariable(n) gesetzt werden. In einer entsprechenden Methode müsste dann ein *Process*⁶ gestartet werden über welchen der Solver angesprochen werden würde. Über entsprechende Input/Output-Streams würde das LP-Modell eingelesen bzw. das Ergebnis zurückgegeben werden. Über die genau Funktionsweise dieses Ansatzes soll auf das Projektteam „Diätplaner“ verwiesen werden.

⁶ Vgl. <http://docs.oracle.com/javase/7/docs/api/java/lang/Process.html>, Abruf: 19.01.2015

Auch wurde „GLOP“ näher analysiert. Eine detailliertere Beschreibung sowie Einbindung in ein Java-Projekt kann unter⁷

Einführung : <https://developers.google.com/optimization/docs/lp/glop>

API : <https://developers.google.com/apps-script/reference/optimization/>

Installation : <https://developers.google.com/optimization/installing>

nachgelesen werden. Wie bereits erwähnt wurde jedoch dieser Ansatz als Notfallplan ausgearbeitet und direkt wieder verworfen, nachdem Ansatz 1 funktionierte.

8.1 Problemerkörterung

Die Applikation JobShopNeu ließ sich bisher über eine Jar-Datei starten, jedoch konnte keine Berechnung durchgeführt werden. Wurde das Projekt in Eclipse eingebunden und gestartet, gab es unter anderem Laufzeitfehler bezüglich der Schnittstelle zu lpSolve (z.B.: no lpsolve55j in java.library.path oder lpSolve55j.dll konnte nicht gefunden werden). Nach einiger Recherche haben wir herausgefunden, dass einige DLL-Dateien für verschiedene Betriebssysteme zwar im Projekt abgelegt wurden, jedoch zur Lauffähigkeit noch weitere fehlten. Im folgenden Abschnitt wird der Zusammenhang zur DLL-Dateien näher erläutert:

DLL ("Dynamic Link Library")-Dateien wie z.B. lpsolve55.dll sind kleine Programme, ähnlich EXE ("ausführbare")-Dateien, die es verschiedenen Softwareprogrammen ermöglichen, dieselbe Funktion zu nutzen (z.B. drucken). Die lpsolve55.dll ist eine DLL-Datei, die im Zusammenhang mit lpSolve steht, dass von Mixed Integer Linear Program Solver für das Windows-Betriebssystem entwickelt wurde. Die aktuellste Version von Lpsolve55.dll ist momentan 5.5.0.10.

DLL Fehler können auftreten, weil sie eben gemeinsam genutzte Dateien sind und so Windows beim Laden dieser auf Fehler und somit auf einen Abbruch der Applikation stoßen kann. Außerdem können DLL-Dateien beschädigt sein.

8.2 Lösungsansatz

Um die vorher angedeuteten Probleme zu beheben, haben wir die aktuellsten DLL-Dateien lpsolve55.dll und lpsolve55j.dll heruntergeladen. Diese können von der Bit-Rechnerarchitektur anschließend direkt in das Projekt eingebunden werden. Somit wurde der Fehler behoben und die Berechnung lief in Eclipse. Anschließend wird das gesamte Projekt in

⁷ Abgerufen am 19.01.2015

einer JAR-Datei gespeichert. Diese kann nun auf Grund der DLL Abhängigkeit jedoch nicht direkt per Doppelklick ausgeführt werden. In nachfolgendem Beispiel für eine 64Bit-Rechnerarchitektur werden die bisherigen Schritte näher erläutert⁸:

8.3 Vorgehensweise zur Erstellung der 64 Bit-Applikation:

Die beiden DLL-Dateien für eine 64Bit-Architektur direkt in das Projekt einbinden, wenn nötig überschreiben:

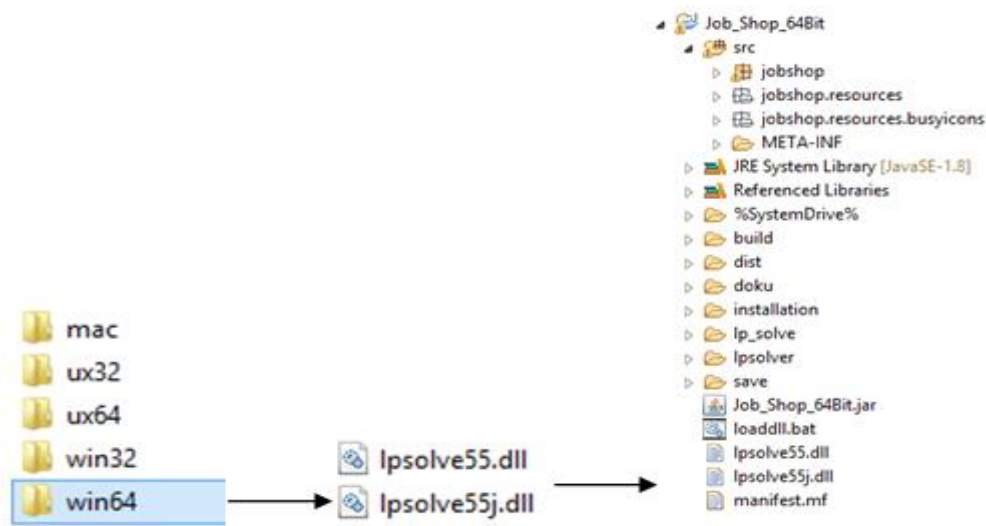


Abbildung 13: DLL in Projekt importieren

Als runnable Jarfile speichern:

- In der Launch configuration die Startklasse auswählen
- In der Export destination den Projektordner auswählen und JAR-Datei benennen (Hier: Job_Shop_64Bit.jar)

⁸ Gilt analog für die Erstellung der 32 Bit-Applikation

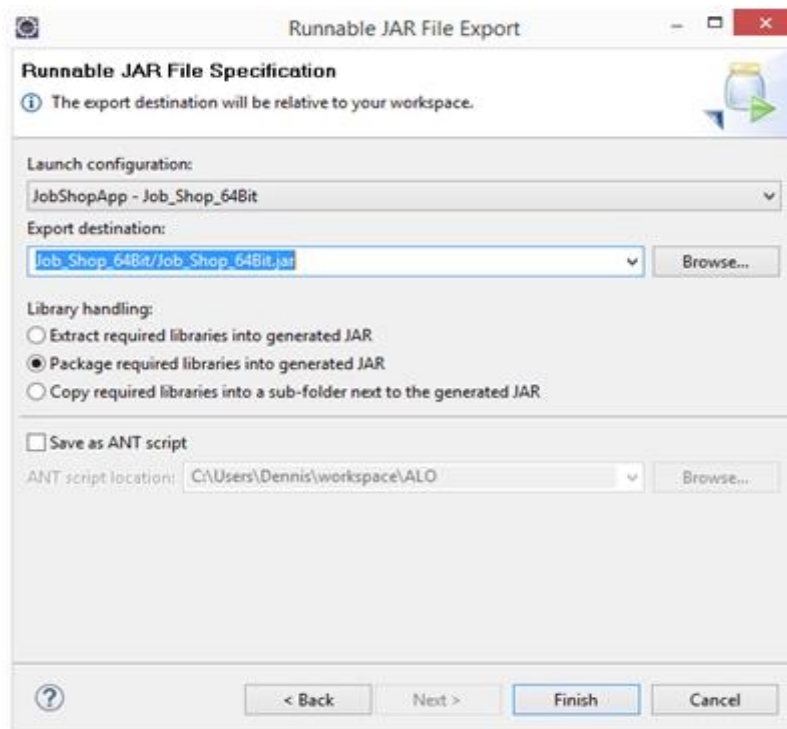


Abbildung 14: Runnable JAR File Export

Damit die JAR-Datei sich starten lässt, wurde direkt im Projektordner eine BATCH-Datei erstellt, welche diese aufruft.

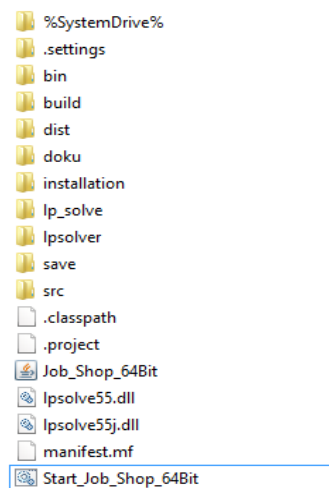


Abbildung 15: Projekttinhalt

Inhalt der Batch-Datei Start_Job_Shop_64Bit:

```

@echo off
echo.
echo.
echo.
echo -----  JobShopNeu 64Bit wird gestartet  -----
echo.
echo.
echo.
java -jar Job_Shop_64Bit.jar
EXIT

```

Abbildung 16: Aufbau Start_Job_Shop_64Bit.bat-Datei

Somit lässt sich die Applikation JobShopNeu über die Batch-Datei starten:

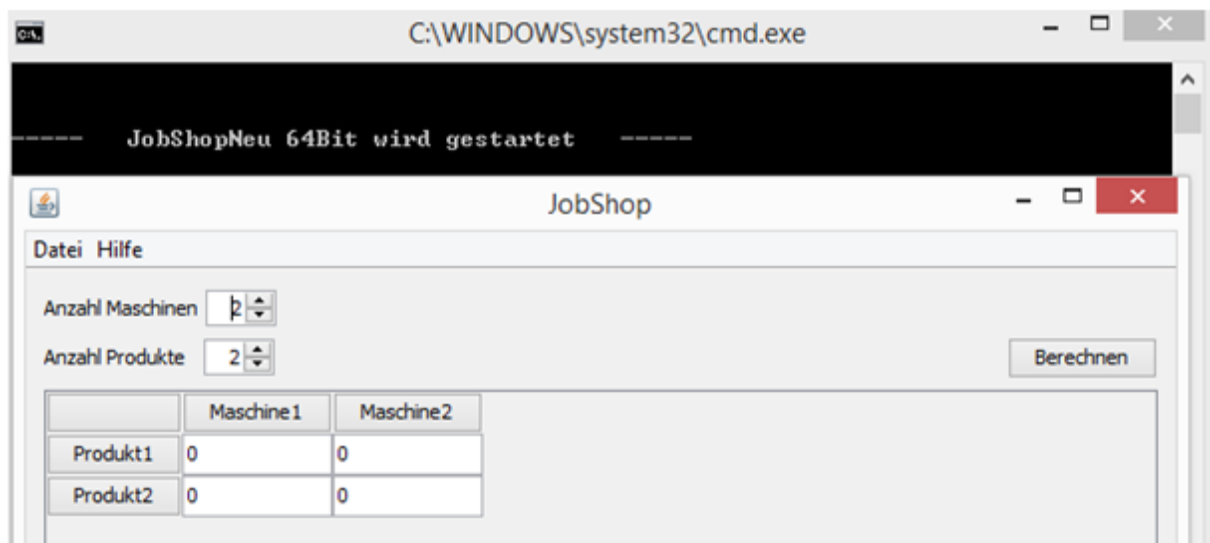


Abbildung 17: Programmaufruf

9 Weiterentwicklung JobShopNeu

Im Folgenden werden alle Aktivitäten zur Erstellung der Lauffähigen JobShopNeu (32-/64-Bit) näher erläutert.⁹

Die Anwendung wurde des Weiteren wie folgt weiterentwickelt:

- Eine Erweiterte Eingabematrix von 16 Maschinen und 16 Produkten

Die Code wurde insofern erweitert, dass nun die Möglichkeit besteht einen Maschinen- und Produktanzahl von 16 auszuwählen und diese berechnen zu lassen. Davor lag die Anzahl bei lediglich 10.

⁹ Zu berücksichtigen sind ebenfalls die im Code enthaltenen Kommentare.

	Maschine1	Maschine2	Maschine3	Maschine4	Maschine5	Maschine6	Maschine7	Maschine8	Maschine9	Maschine10	Maschine11	Maschine12	Maschine13	Maschine14	Maschine15	Maschine16
Produkt1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Produkt2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Produkt3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Produkt4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Produkt5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Produkt6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Produkt7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Produkt8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Produkt9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Produkt10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Produkt11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Produkt12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Produkt13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Produkt14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Produkt15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Produkt16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Abbildung 18: Eingabematrix 16/16

Dabei mussten die Maximalwerte der Methode *initComponents()* in der Klasse *JobShopView* auf 16 gesetzt werden:

```
private void initComponents() {

    mainPanel = new JPanel();
    jLabel1 = new JLabel();
    jLabel2 = new JLabel();
    spinMod1 = new SpinnerNumberModel(2, 1, 16, 1);
    jSpinner1 = new JSpinner(spinMod1);
    spinMod2 = new SpinnerNumberModel(2, 2, 16, 1);
}
```

Abbildung 19: Methode initComponents()

- Erstellung eines PopUp Fensters bei Falscheingabe (Bsp.: String)

Um Falscheingaben hinsichtlich der Durchlaufzeit zu vermeiden, wurde die Methode *stringToValue(String s)* aus der Klasse *JobShopNumberFormatter* insofern erweitert, dass bei Eingabe eines Strings ein PopUp Fenster mit der Meldung „Sie müssen eine Ganzzahl eingeben“ erscheint.

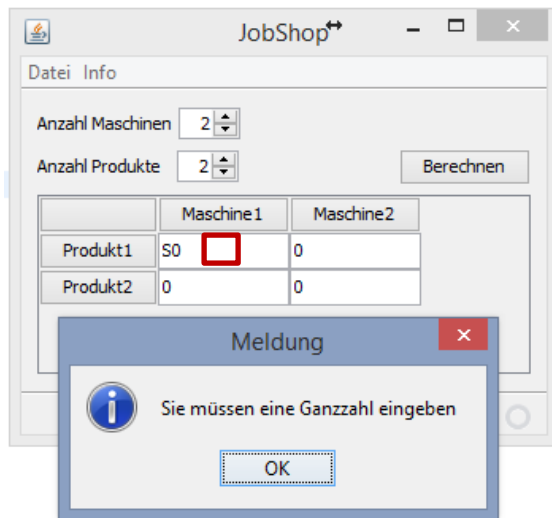


Abbildung 20: Popup Beispiel bei Falscheingabe

Sofern eine Ganzzahl über 99 eingegeben wird, springt diese automatisch auf 0, da nur Werte bis 99 akzeptiert werden.

```
public Object stringValue(String s) throws ParseException {
    Number number = null;
    if(s.matches("\\d*")) {
        if(s.length() > 2) {
            number = 99;
        } else {
            number = (Number)super.stringValue(s);
        }
    } else {
        JOptionPane.showMessageDialog(null, "Sie müssen eine Ganzzahl eingeben");
        number = 0;
    }
    return number;
}
```

Abbildung 21: Methode stringValue(String s)

- Das zu berechnende LP Modell (Eingabematrix) wird gespeichert/ geladen

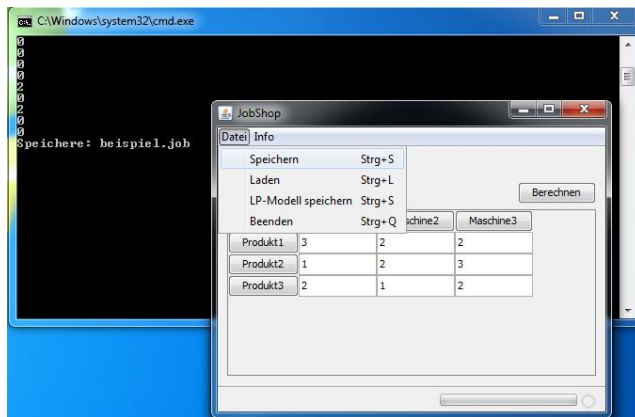


Abbildung 22: Eingabematrix speichern

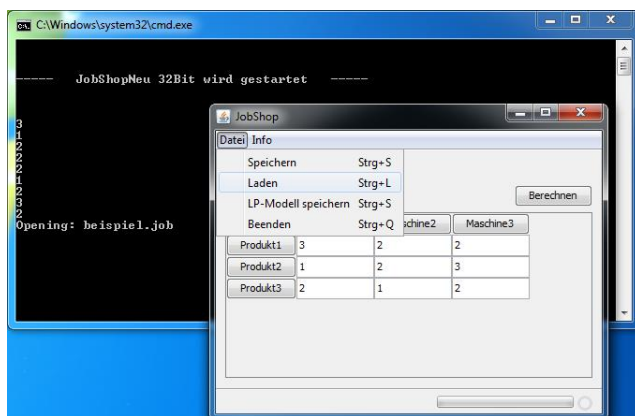


Abbildung 23: Eingabematrix laden

- Speichern des aufgestellten LP-Modells sowie den Lösungsvektor

Zur Überprüfung der Fachlichkeit wurde der Code um einen FileWriter ergänzt, welcher das erstellte LP-Modell in einer temporären TXT-Datei abspeichert. Ebenfalls wird eine temporäre TXT-Datei für die Speicherung der berechneten Lösung im entsprechenden Projektverzeichnis erstellt.

```

300     try {
301         file = File.createTempFile("LP-Modell", ".txt", new File(dirName));
302         writer = new FileWriter(file);
303         for (Map<String, String> map : matrix) {
304             writer.write(map.get("function"));
305             writer.write(map.get("operator") + this.separator);
306             writer.write(map.get("result"));
307             writer.write("\r\n");
308         }
309         writer.flush();
310         writer.close();
311     } catch (IOException e) {
312         e.printStackTrace();
313         System.out.println("File konnte nicht erstellt werden!");
314     }
315
316     /* Save the solution from lpsolve in a separate file in the project directory.
317     * Following code extended at 12.01.2015 by Sebastian Stephan
318     */
319     try {
320         file2 = File.createTempFile("LP-Solve", ".txt", new File(dirName));
321         solver.setOutputfile(file2.getName());
322         solver.printObjective(); // print lpsolve solution (output) in command box, text-file and cmd
323         solver.printSolution(1); // print lpsolve solution (output) in command box, text-file and cmd
324         solver.printConstraints(1); // print lpsolve solution (output) in command box, text-file and cmd
325     } catch (IOException e) {
326         e.printStackTrace();
327         System.out.println("File konnte nicht erstellt werden!");
328     }
329
330     file.deleteOnExit();
331     file2.deleteOnExit();
332

```

Abbildung 24: FileWriter-Erweiterung

Der Lösungsvektor für das mitgelieferte Beispiel-LP-Modell sieht dann z.B. wie folgt aus:

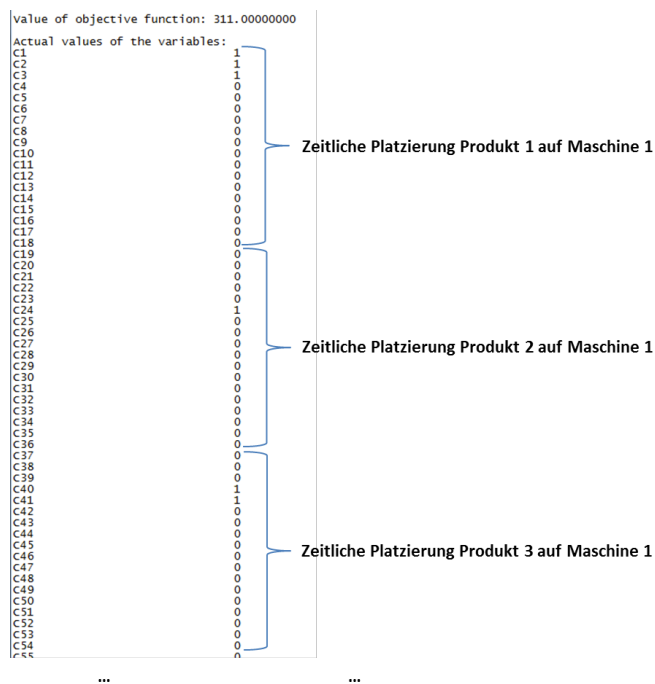


Abbildung 25: Lösungsvektor lpSolve

10 Beispielaufgabe

Im Folgenden soll die Funktionsweise der Software anhand einer kleinen Beispielaufgabe veranschaulicht werden.

Auf 3 Maschinen können jeweils 3 Produkte bearbeitet werden. Jedes Produkt kann pro Zeiteinheit (ZE) nur von genau einer Maschine bearbeitet werden. Ebenso kann eine Maschine immer nur ein Produkt pro ZE bearbeiten. Die jeweiligen Durchlaufzeiten der Produkte auf den Maschinen sind in nachfolgender Tabelle aufgezeigt:

	Maschine1	Maschine2	Maschine3
Produkt1	3	2	2
Produkt2	1	2	3
Produkt3	2	1	2

Aufgabe: Erstellen Sie einen Maschinenbelegungsplan bei dem die Durchlaufzeit möglichst minimal ist (optimal).

10.1 Lösung traditionell mit IpSolve:

Zunächst müsste ein LP-Modell erstellt werden. Dieses Modell würde dabei unter Berücksichtigung der entsprechenden Anforderungen

- Ein Produkt kann pro Zeitintervall nur auf einer Maschine bearbeitet werden
- Eine Maschine kann nur an einem Produkt pro Zeitintervall arbeiten
- Produkte sollen möglichst am Stück bearbeitet werden (Zusatz)

über >300 Variablen sowie über >500 Restriktionen umfassen. Dieses LP-Modell aufzustellen würde sehr aufwändig sein. Aus diesem Grund wurde die Software "JobShop" entwickelt.

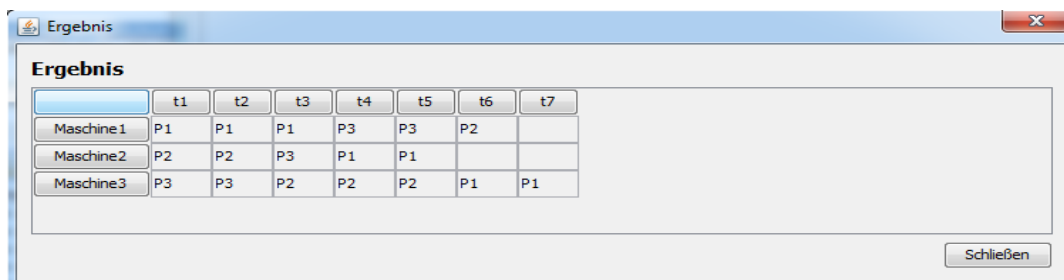
10.2 Lösung mit JobShopNeu:

Die Eingabe erfolgt in Form einer Matrix.

	Maschine1	Maschine2	Maschine3
Produkt1	3	2	2
Produkt2	1	2	3
Produkt3	2	1	2

Abbildung 26: Eingabematrix JobShopNeu

Entsprechend den genannten Anforderungen wird folgender Lösungsvektor berechnet:



The screenshot shows a window titled 'Ergebnis' with a table of the solution matrix. The table has rows for 'Maschine1', 'Maschine2', and 'Maschine3' and columns for time intervals 't1' through 't7'. The cells contain product identifiers (P1, P2, P3) indicating which product is being processed on which machine during which time interval.

	t1	t2	t3	t4	t5	t6	t7
Maschine1	P1	P1	P1	P3	P3	P2	
Maschine2	P2	P2	P3	P1	P1		
Maschine3	P3	P3	P2	P2	P2	P1	P1

Abbildung 22: Lösungsmatrix JobShopNeu

Hinweis:

Fertigstellungszeiträume einzelner Produkte werden nicht im LP-Modell mit berücksichtigt.

Bug:

Wird ein gespeichertes LP-Modell geladen, müssen zuerst alle Texteingabefelder „aktiviert“ werden, d.h. durch einen Mausklick in entsprechendes Feld oder Ähnlichem. Grund hierfür liegt in der verwendeten Softwarekomponente „SingleFrameApplication“

11 Fazit

Die Lehrveranstaltung Anwendung der Linearen Optimierung (ALO) ermöglichte uns erste Einblicke in die Planung, Modellierung und Implementierung von anwenderfreundlichen Problemlösungen auf der Grundlage von Modellen der Linearen Programmierung (Operations Research). Dabei lag der Fokus nicht nur ausschließlich auf der fachlichen Ebene, der Entwicklung mathematisches LP-Modell, sondern auch deren Lösung softwaretechnisch durch Eigenimplementierung sowie unterschiedlicher LP-Solver-Produkte (z.B. lpSolve, GLOP). Durch die Bearbeitung verschiedener Problemstellungen in Form von Projekten konnten wir bereits gelerntes direkt anwenden und so weitere praktische Erfahrungen sammeln. Im Team aber auch durch Projektbesprechungen zusammen mit Prof. Grütz lernten wir für auftauchende Problemstellungen Lösungsskizzen und –alternativen auszuarbeiten und ggf. umzusetzen.

Einziger Kritikpunkt sind die zum Teil hohen projektabhängigen softwaretechnischen Anforderungen.

Zusammenfassend lässt sich festhalten, dass die Projektarbeit im Rahmen der Veranstaltung ALO sehr interessant war. Es wurden Einblicke in die praktische Anwendung der Linearen Optimierung ermöglicht sowie die OR-Kenntnisse gefestigt und erweitert. Außerdem konnten wir, da es sich um ein Java-Projekt handelte, zusätzlich unsere Programmier- und Softwaremodellierungskenntnisse erweitern.