Benjamin Franz MAKULA, BSc.

Enterprise Management Software Framework
Tarife als Basis von Verrechnungsprozessen

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

Masterstudium Informatik

eingereicht an der

Technischen Universität Graz

Betreuer

Univ.-Prof. Dipl.-Ing. Dr. techn. Wolfgang SLANY

Institut für Softwaretechnologie

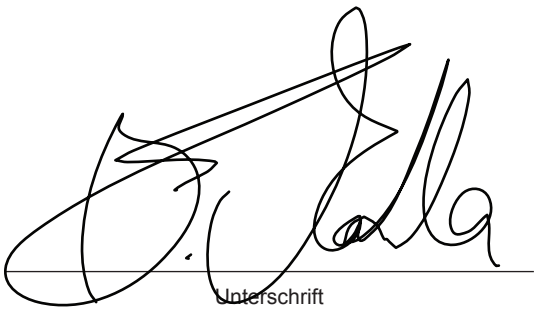
Klagenfurt am Wörthersee, August 2017

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

30.08.2017

Datum



Unterschrift

Danksagung

>> Keine Schuld ist dringender, als die, Dank zu sagen <<

Marcus Tullius Cicero

In diesem Sinne möchte ich mich bei all jenen universitären und privaten Wegbegleitern bedanken, die mich in der Zeit des Schaffens unterstützt und gefördert haben. Hervorheben möchte ich aber in diesem Zusammenhang die verständnisvolle Supervision des betreuenden Professors in Abschnitten der Stagnation, die mitunter die ausschlaggebende Kraft war, alle aufgetretenen unscharfen Ziele neu zu ordnen.

Ganz besonders möchte ich mich aber bei meiner Familie und meiner zukünftigen Frau für das jahrelange Vertrauen in mich bedanken. Ohne dem gebotenen Rückhalt und die vermittelte Geborgenheit in manchen Zeiten der Unwirklichkeit, wäre diese Arbeit wohl nie vollendet worden. Somit ist diese Arbeit auch ein Verdienst eures Wirkens und soll auch euch gewidmet sein.

Kurzfassung

In der heutigen Zeit sehen sich vermehrt kleine- und mittelständige Unternehmen (KMU) mit dem Problem eines nicht zeitgemäßen Verrechnungsmanagements konfrontiert. Häufig werden manuelle Verrechnungsprozesse angewendet, die mit einer personellen Bindung einhergehen.

Dienstleistungsunternehmer mit Serviceaufträgen von Fremdunternehmen sind zudem noch mit externen Verrechnungsschemen konfrontiert, welche in Servicetarifen verankert sind. Die Mannigfaltigkeit solcher Tarife und dessen Komplexität bringen ein Mehr an Aufwand, sowie eine noch höhere personelle Bindung an versiertem Personal. Diese Gründe, sowie finanzrechtliche Änderungen sind mitunter eine treibende Komponenten für eine Steigerung der Nachfrage für Softwarelösungen in diesem Bereich.

In dieser Arbeit wird nun aus aktueller technischer Sicht beleuchtet, welches Ausmaß ein solches Softwaresystem für KMU haben sollte und welche technischen Mittel sich für eine praktische Umsetzung eignen. Getroffene Architektur- und Entwurfsentscheidungen werden auf ihre Sinnhaftigkeit hinterfragt und mittels grundlegender Entwurfs- bzw. Architekturmuster begründet. Im konkreten werden Technologien rund um die Programmiersprache Java herangezogen, um ein konzeptionelles Programmframework zu entwickeln, das durch passende Architekturmuster sehr einfach adaptierbar ist und sich somit für eine breite Zahl an KMU eignet.

Des Weiteren wird auf die bereits erwähnten Tarife eingegangen und analysiert, aus welchen elementaren Teilen sie zusammengesetzt sind. Hierfür wird auch ein formales mathematisches Modell eingeführt und damit ermittelt, inwieweit Tarife abstrahierbar sind, damit sie eine maximale Flexibilität in Bezug auf Konfigurierbarkeit und Adaptierung bieten können.

Abstract

Today more and more small and medium-sized enterprises (SMEs) face the challenge to improve and modernize their billing management processes. In many cases of these, companies use the common way of a manually handled billing process which goes along with a higher personnel binding.

In case of service companies with service orders of third-party enterprises, the billing process also has to handle the external billing schemes which are defined in custom tariffs. The diversity and complexity of these tariffs implicates rising operating expenditures and an increasing personnel binding. In addition to these reasons, changes in the Austrian financial law pushes the demand on software based solutions for these problems.

In the course of this thesis, solutions to above mentioned questions are worked out with special focus on the technical fundamental principles which a software based solutions for SMEs should be based on. The recommended technical fundamentals which are selected in this thesis are going to be assessed and justified with common used design patterns. This thesis will evaluate the marginal conditions of a conceptual program framework written in Java. The framework is based on carefully selected and well-chosen design patterns which implicate a high reusability factor for different types of SMEs.

A detailed evaluation of tariffs including their fundamentals is part of this thesis as well. For this purpose a Formal mathematical model is developed. Based on this model the tariffs' abstractability with respect to flexibility and configurability is assessed.

Inhaltsverzeichnis

Danksagung iii

Kurzfassung iv

Inhaltsverzeichnis vi

1. Einleitung 8

1.1 Kleine und mittlere Unternehmen 9
1.2 Einzelaufzeichnungs-, Registrierkassen- und Belegerteilungspflicht 10
1.3 Empirie zu automatisierten Geschäftsprozessen 12
1.3.1 Fragestellung 13
1.3.2 Methodik und Durchführung 14
1.3.3 Zusammenfassung 18

2. Tarifproblematik 19

2.1 Tarifanalyse 20
2.1.1 Tarifmodell 20
2.1.2 Tariflogik 22
2.1.3 Tariftransformation 23
2.2 Recommender Systems für Tarife 26
2.2.1 Was ist eigentlich eine Tarifempfehlung 27
2.2.2 UC1 – Die Tarifempfehlung als Vektorenoptimierungsproblem 29
2.2.3 UC2 – Tarifempfehlung mit variablem Eingabevektor 35
2.2.4 Zusammenfassung 50
2.3 Tarife in Hochsprachen 51
2.3.1 Schlüsselprobleme 52
2.3.2 Objektorientierte Umsetzung 53
2.3.3 Zusammenfassung 77

3. Java EE Framework mit Tarifintegration 79

3.1 Randbedingungen 79
3.2 Designentscheidungen 80
3.2.1 Client/Server Architektur 80

3.2.2 Java Enterprise Edition 81
3.2.3 JPA, EJB, JSF oder auch MVC – Pattern 82
3.3 Frameworkarchitektur 84
3.3.1 Gesamt Architektur 85
3.3.2 JavaRates Integration 86

4. Zusammenfassung 93

5. Literaturverzeichnis 95

6. Abbildungsverzeichnis 97

7. Anhang 98

1. Einleitung

Der Einzug von Softwaresystemen als Lösungsansatz aktueller Problemstellungen ist omnipräsent. Nicht nur in privaten Bereichen findet man immer mehr Softwarelösungen, die eine Effizienzsteigerung in nahezu jeder Lebenslage versprechen. Auch im Unternehmensbereich erkennt man einen Anstieg an Softwaresystemen, die Geschäftsprozesse abbilden, vereinfachen, aber auch optimieren sollten. [1]

In dieser Arbeit wird nun die Implementationsfähigkeit von Software in Geschäftsprozessen analysiert. Primär wird auf Geschäftsprozesse im Bereich des Verrechnungsmanagements eingegangen, welche mitunter eine starke Bindung zu s.g. Tarifen aufweisen. Da sich das Themenfeld des Verrechnungsmanagements mit Bereichen des ERP(Enterprise-Resource-Planning) und des Kundenbeziehungsmanagements überschneiden, werden auch diese Bereiche zur Sprache kommen. Konkretisiert wird diese Arbeit durch das Setzen des Hauptaugenmerkes auf kleine und mittlere Unternehmen.

Angesichts dieser Rahmenbedingungen sollte daher anfänglich definiert werden, was kleine und mittlere Unternehmen sind und welche wirtschaftliche Rolle sie einnehmen. Fundiert werden diese Aussagen durch Statistiken aus dem österreichischen Wirtschaftsraum.

Des Weiteren wird in der Einleitung auf die Notwendigkeit von Softwaresystemen hinsichtlich neuer Gesetzgebungen eingegangen. Hierfür wird speziell auf den Erlass „BMF-010102/0012-IV/2/2015“ des österreichischen Finanzministeriums eingegangen, der die „Einzelaufzeichnungs-, Registrierkassen- und Belegerteilungspflicht“ in Österreich neu reglementiert.

Abgeschlossen wird die Einleitung durch eine Empirie bezüglich des Einsatzes von Softwarelösungen für Geschäftsprozesse von kleinen und mittleren Unternehmen. Hinsichtlich der aktuellen Unternehmensverteilung nach Beschäftig-

[1] Forbes - http://www.forbes.com/sites/louiscolumbus/2013/06/18/gartner-predicts-crm-will-be-a-36b-market-by-2017/#7cb8e081c1ec – 03.08.2016 – Anhang 1

tengrößenklassen wird das Augenmerk auf Klein- und Kleinstunternehmen geworfen.

1.1 Kleine und mittlere Unternehmen

Bei der Kategorisierung von Unternehmen nach Beschäftigungsgrößen findet man eine Unzahl an Klassifikationen. So gibt es sowohl internationale als auch nationale Klassifikationen, die als Grundlage statistischer Erhebungen fungieren. Durch eine Empfehlung[2] der europäischen Union sollte eine europaweite Vereinheitlichung eingeleitet werden. Basierend auf dieser Empfehlung wird nicht nur die Beschäftigungsgröße berücksichtigt, sondern auch der Umsatz, die Bilanzsumme und die Eigenständigkeit eines Unternehmens.

Diese Kategorisierung(siehe Tabelle 1) bildet nun einen ersten Ausgangspunkt für die weiteren Analysen, die mit Hilfe statistischer Erhebungen der Statistik Austria durchgeführt werden. Im Konkreten wird die „Leistungs- und Strukturstatistik“[3] von „Produktion & Dienstleistungen“ aus dem Jahr 2013 analysiert und die Rolle von kleinen und mittleren Unternehmen erarbeitet.

Da diese Statistik nur eine Differenzierung von Beschäftigungsgrößen- und Umsatzgrößenklassen verwendet und keine erweiterte Kategorisierung nach 2003/361/EG vornimmt, wird diese Analyse nur anhand dieser Unterscheidungskriterien durchgeführt.

Demgemäß können folgende Informationen aus dieser Statistik entnommen werden. Im Kalenderjahr 2013 konnte Österreich **324.709** Unternehmen in den Bereichen Produktion und Dienstleistungen aufweisen. Diese Unternehmen konnten einen Umsatzerlös in Höhe von **709,5 Mrd. €** erwirtschaften und stellten ein **39,2 Mrd. €** Investitionsvolumen (IV). Aus dieser Statistik kann man auch entnehmen, dass ca. **99%** der österreichischen Unternehmer der Produk-

[2] Empfehlung der Kommission vom 6. Mai 2003 betreffend die Definition der Kleinstunternehmen sowie der kleinen und mittleren Unternehmen - K(2003) 1422 - 2003/361/EG

[3] STATISTIK AUSTRIA, „LEISTUNGS- UND STRUKTURSTATISTIK Produktion & Dienstleistungen“, 2015, ISBN 978-3-902925-77-0

tions- und Dienstleistungssparten kleine und mittlere Unternehmen sind. Des Weiteren geht auch hervor, dass **98%** der Unternehmen Klein- und Kleinstunternehmen sind. Diese sind sowohl mit **36,2%** am Gesamtumsatzerlös, als auch mit **35,3%** am Investitionsvolumen beteiligt (siehe Tabelle 2). **[1]**

Tabelle 1: Kategorisierung von Unternehmen nach 2003/361/EG

	Mitarbeiter	**Umsatz**	**Bilanzsumme**	**Eigenständigkeit**
Kleinstunternehmen	bis 9	≤ 2 Mio EUR	≤ 2 Mio EUR	< 25% [4]
Kleinunternehmen	10 bis 49	≤ 10 Mio EUR	≤ 10 Mio EUR	
Mittlere Unternehmen	50 bis 249	≤ 50 Mio EUR	≤ 43 Mio EUR	
Großunternehmen	ab 250	> 50 Mio EUR	> 43 Mio EUR	

Angesichts dieser Zahlen, sollte nun die wirtschaftliche Rolle der kleinen und mittleren Unternehmen klar ersichtlich sein. Hinsichtlich des folgenden Kapitels zum Thema „Empirie zu automatisierten Geschäftsprozessen“ werden diese Werte auch die Grundlage bilden, um die passenden Parameter für eine aussagekräftige Umfrage zu finden.

Tabelle 2: Prozentuelle Verteilung der Ergebnisse nach Beschäftigtengrößenklassen[4]

	Mitarbeiter	**Anzahl**	**Umsatzerlös**	**IV**
Kleinstunternehmen	bis 9	87,3 %	16,8 %	18,1 %
Kleinunternehmen	10 bis 49	10,7 %	19,4 %	17,2 %
Mittlere Unternehmen	50 bis 249	1,7 %	27,6 %	21,6 %
Großunternehmen	ab 250	0,3 %	36,2 %	43,1 %

1.2 Einzelaufzeichnungs-, Registrierkassen- und Belegerteilungspflicht

Wie bereits anfänglich erwähnt, befasst sich diese Arbeit auch mit Softwarelösungen für Geschäftsprozesse im Bereich des Verrechnungsmanagements. Da die Waren- bzw. Leistungsverrechnung einer staatlichen Normierung unterliegt, ist es zunächst wichtig auch die rechtliche Lage zu kennen. Das Rechnungswesen richtet sich beim Teilbereich der Rechnungslegung unter anderem auch an das Umsatzsteuergesetz (§ 11 UStG, BGBl. Nr. 663/1994) und an die Bundes-

[4] Kapitalanteile/Stimmrechte im Fremdbesitz < 25 Prozent (siehe 2003/361/EG – Anhang - TITEL I - Artikel 3)

abgabenordnung (§ 131 BAO, Bundesabgabenordnung, BGBl. Nr. 194/1961). Da das Recht im Grunde einer stetigen Veränderung unterliegt, gibt es auch im Bereich der Rechnungserstellung immer wieder rechtliche Anpassungen[5]. Hinsichtlich der aktuellen Anpassungen sollte eingangs erwähnt werden, dass diese Gesetzesänderungen durchaus auch als Katalysator für die Steigerung der Nutzung von Software im Bereich des Verrechnungsmanagements gesehen werden können, was man sehr gut an der medialen Berichterstattung mitverfolgen kann (siehe Anhang 2-5). [6] [7] [8] [9]

Im konkreten geht es bei diesen rechtlichen Anpassungen um die s.g. „Einzelaufzeichnungs-, Registrierkassen- und Belegerteilungspflicht“[10]. Bei diesem Erlass des Finanzministeriums geht es um die gesetzliche Verpflichtung Rechnungen über Bargeldverkehr elektronisch zu erzeugen und anschließen manipulationssicher zu speichern. Die gesetzliche Grundlage findet man in der Bundesabgabenordnung.

„Betriebe haben alle Bareinnahmen zum Zweck der Losungsermittlung mit elektronischer Registrierkasse, Kassensystem oder sonstigem elektronischen Aufzeichnungssystem unter Beachtung der Grundsätze des § 131 Abs. 1 Z 6 einzeln zu erfassen.“ (§ 131b (1) BAO, Bundesabgabenordnung, BGBl. Nr. 194/1961)

Da es sich beim oben beschriebenen Erlass nur um eine Regelung für Bargeldverkehr handelt, sind vorerst nur bestimmte Unternehmen verpflichtet ihre Verrechnungsprozesse anzupassen. Dennoch kann man aus solch einer Gesetzesänderung durchaus folgern, dass es in absehbarer Zeit mitunter möglich sein könnte, dass es eine generelle Regelung hin zu manipulationsgeschützten elektronischen Rechnungen geben könnte.

---

[5] Bundeskanzleramt der Republik Österreich, RIS, Gesamte Rechtsvorschrift für Umsatzsteuergesetz 1994 – 56 Änderungen seit 1994 – Stand 05.06.2016

[6] Wiener Zeitung – 16.03.2015 – Registrierkassen-Absatz könnte um bis zu 10 Prozent zulegen

[7] Kurier – 27.03.2015 – Die Renaissance der Registrierkasse

[8] Kleine Zeitung – 19.03.2015 – Registrierkassenpflicht: Gutes Geschäft für Hersteller

[9] Der Standard – 02.03.2016 – Registrierkassenmesse: "Es trifft immer die Kleinen"

[10] Erlass des BMF vom 12.11.2015, BMF-010102/0012-IV/2/2015, BMF-AV Nr. 169/2015

Bei einer Hinterfragung der aktuellen Gesetzgebung hinsichtlich des Rechnungslegungsverfahren (§ 11 UStG, BGBl. Nr. 663/1994) kann man feststellen, dass eine Kontrolle durch die Finanzkontrollbehörden nur durch die Aufbewahrungspflicht sowie die auf Rechnungen vermerkten Kundenanschriften gewehrleistet werden kann. Eine manuelle Prüfung ist daher mit Umständen verbunden. Manipulationsgeschützte elektronische Rechnungssysteme, angelehnt am Registrierkassensystem, würden hinsichtlich finanzrechtlicher Kontrollen womöglich zu einer Steigerung der Effizienz seitens der Behörden führen.

Eine solche Gesetzesänderung könnte unter dem gleichen Gesichtspunkt forciert werden, wie es bei der aktuellen Regelung gemacht wurde. Die Registrierkassen- und Belegerteilungspflicht sollte im Sinne der Behörden eine Manipulation des wahren Bargeldverkehrs hin zu einem steuerschonenden Bargeldverkehr verhindern (§ 1 Z 1 RKSV, Registrierkassensicherheitsverordnung, BGBl. II Nr. 410/2015).

Sowohl die Gesetzeslage als auch die allgemein wahrnehmbare und fortschreitende Technologisierung von Unternehmen, legitimiert daher Analysen hinsichtlich der technischen Umsetzbarkeit von Softwaresystemen in den Bereichen des ERP(Enterprise-Resource-Planning) und des Verrechnungsmanagements. Die in dieser Arbeit durchgeführten Analysen bilden anschließend die Grundlage für ein konzeptionelles Framework, welches als „best practice“ Konzept für Softwaresysteme in diesen Bereichen dienen sollte.

1.3 Empirie zu automatisierten Geschäftsprozessen

Im vorangegangenen Kapitel wurde die rechtliche Lage hinsichtlich Rechnungslegung dargestellt und eine Bewusstseinsbildung hinsichtlich möglicher zukünftiger Änderungen im Rechnungswesen anhand von Änderungen im UStG gefördert.

Bezugnehmend auf vorangegangene Erkenntnisse stellt sich nun die Frage, in wieweit es bei kleinen und mittleren Unternehmen bereits ein elektronisches Rechnungssystem gibt, bzw. mit welchen Mitteln die Rechnungslegung aktuell durchgeführt wird. Allgemeine Beobachtungen zeigen, dass ein erheblicher An-

teil von Klein- und Kleinstunternehmern auf Textverarbeitungs-Software setzen und auch ihren Verrechnungsprozess auf diese aufbauen.

Dieser Abschnitt sollte ergänzend zu den angeführten Punkten im Kapitel 1.2 die in dieser Arbeit durgeführten Analysen legitimieren. Da es sich bei dieser Arbeit nicht nur um eine auf Umfragen basierte empirische Forschungsarbeit handelt, fällt dieser Abschnitt nur rudimentär aus und konzentriert sich ausschließlich auf die wesentlichen Fragen, die eine Legitimierung unterstreichen.

1.3.1 Fragestellung

Im Zuge dieser Empirie wird auf folgende Fragestellung eingegangen:

„Wie viel Prozent der Klein- und Kleinstunternehmen verwenden eine Softwarelösung zur Erstellung und Verwaltung Ihrer Rechnungen?“

Um diese Fragestellung zu konkretisieren, muss man noch alle Randbedingungen definieren.

1. Klein- und Kleinstunternehmen kategorisiert nach 2003/361/EG (siehe Kapitel 1.1)

2. Die betroffenen Unternehmen befinden sich im österreichischen Wirtschaftsraum

3. Als Softwarelösung werden nur Anwendungen bezeichnet, die sich durch Ihre Funktionalität auf Rechnungserstellung spezialisiert haben. Folglich werden Textverarbeitungs- bzw. Tabellenkalkulationsprogramme und ähnliche multifunktionale Anwendungen nicht als Softwarelösungen hinsichtlich der Fragestellung geführt.

4. Mit dem Erstellen von Rechnungen ist der Prozess der strukturierten Eingabe hinzu einer standardisierten Rechnung gemäß (§ 11 UStG, BGBl. Nr. 663/1994) gemeint[11].

1.3.2 Methodik und Durchführung

Um die im vorangegangenen Kapitel beschriebene Frage beantworten zu können, wurden nun folgende Überlegungen getätigt.

Da es sich bei dieser Empirie um ein Abbild der aktuellen Lage von Klein- und Kleinstunternehmen handeln muss, ist eine Befragung dieser Unternehmen Voraussetzung für alle weiteren Erkenntnisse. Im Zuge dieser Arbeit wurde daher eine Stichprobenbefragung durchgeführt, die Aufschluss über die oben genannten Fragen liefern sollte.

Um ein repräsentatives Ergebnis liefern zu können, darf eine Mindestanzahl von Befragten nicht unterschritten werden. Die konkrete Anzahl wurde unter Zuhilfenahme der Formel 2 nach W. G. Cochran berechnet **[2].** Als Parameter wurde eine approximierte Grundgesamtheit von 330.000 Kandidaten (siehe Kapitel 1.1) genommen. Da diese Umfrage nur einen unterstützenden Aspekt haben sollte, wurde ein Konfidenzniveau von 97,5% und Fehlerbereich von 10% definiert.

Formel 1: Berechnungsformel für Konfidenzintervalle einer Normalverteilung[12]

$$\Phi_{0;1}(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-\frac{1}{2}t^2} dt$$

Das Konfidenzintervall von 97,5% entspricht einem approximierten z-*Wert* von 1,96 (siehe Formel 1). Da wir bezüglich dieser Fragestellung kein a priori Wissen hatten, gingen wir von einer Normalverteilung ohne Tendenzen aus ($p = 0.5$).

[11] Zusammengefasst – BMF – „Vorsteuerabzug/Vorschriftsgemäße Rechnung“ – 08.02.2017 www.bmf.gv.at/steuern/selbststaendige-unternehmer/umsatzsteuer/ust-vorsteuerabzug.html

[12] Seite 14 - Cochran, William Gemmell. "Sampling techniques-3." (1977).

Formel 2: Formel zu Ermittlung eines Stichprobenumfangs[13]

$$s = \frac{\frac{z^2 \times p(1-p)}{e^2}}{1 + \left(\frac{z^2 \times p(1-p) \times e^2}{e^2 N}\right)}$$

N … $Grundgesamtheit$ z … $Konfidenzniveau$ e … $Fehlerbereich$

p … $Wahrscheinlichkeitsschätzer$ s … $Stichprobenumfang$

Durch das Einsetzen der konkreten Werte in die Formel 2 erhalten wir folgendes Ergebnis (siehe Formel 3):

Formel 3: Einsetzen der Parameter in Formel 2

$$N = 330000 \qquad z = 1{,}96 \qquad e = 0{,}1 \qquad p = 0{,}5$$

$$s = \frac{\frac{1{,}96^2 \times 0{,}5 \times 0{,}5}{0{,}1^2}}{1 + \left(\frac{1{,}96^2 \times 0{,}5 \times 0{,}5 \times 0{,}1^2}{0{,}1^2 \times 330000}\right)}$$

$$s = 96{,}01205758$$

Basierend auf die oben durchgeführten Berechnungen mussten daher für ein repräsentatives Ergebnis mindestens 96 Klein- und Kleinstunternehmen befragt werden.

Für die Durchführung dieser Umfrage wurde das Onlinebefragungsportal „Umfrage Online“[14] verwendet. Unter einem generierten Weblink konnte die Fragestellung beantwortet werden. Die Unternehmen wurden zufällig ausgewählt und telefonisch bzw. via E-Mail auf die laufende Umfrage hingewiesen. Im Falle einer telefonischen Befragung wurde das Onlinefrageportal vom Fragesteller betätigt. Der Zeitraum der Befragung erstreckte sich von März 2015 bis Juni 2016

[13] Formel 4.1 - Cochran, William Gemmell. "Sampling techniques-3." (1977).

[14] www.umfrageonline.com

Die Umfrage beinhaltete folgende Fragen:

F1. *Wie viele Angestellte hat Ihr/das Unternehmen (<9 oder 10-49)?*

F2. *Verwendet Ihr/das Unternehmen eine Softwarelösung für das Erstellen und Verwalten von Rechnungen (Ja/Nein)?*

(Textverarbeitungsprogramme wie zum Beispiel Microsoft Office(Word, Excel,…) oder Ähnliches zählen als Nein!)

1.3.2.1 *Beschreibung der Stichprobe*

Befragt wurden 98 Klein- und Kleinstunternehmen mit einem regionalen Schwerpunkt von Steiermark, Kärnten und Wien. Die befragten Unternehmen wurden mit Hilfe der Frage F1. in zwei Kategorien eingeteilt.

- UK1 – Kleinstunternehmer
- UK2 – Kleinunternehmer

In der Kategorie UK1 wurden 73 Unternehmen befragt. Die Kategorie UK2 beinhaltete 25 befragte Unternehmen. Anhand dieser Kategorisierung konnten nun folgende drei Ergebnisse hinsichtlich der Frage F2. ermittelt werden.

1. In der Kategorie UK1 besaßen **40.00%** der Unternehmer eine Softwarelösung zu Erstellung und Verwaltung Ihrer Rechnungen
2. In der Kategorie UK2 besaßen **95.00%** der Unternehmer eine Softwarelösung zu Erstellung und Verwaltung Ihrer Rechnungen
3. In beiden Kategorien UK1 und UK2 besaßen **55.00%** der Unternehmer eine Softwarelösung zu Erstellung und Verwaltung Ihrer Rechnungen

Hinsichtlich des dritten Ergebnisses musste noch eine Anpassung hin zur realen Unternehmensverteilung des österreichischen Wirtschaftsraums durchgeführt werden. Demgemäß sollte die Unternehmenskategorie UK1 89,08% und die Unternehmenskategorie UK2 10,92% des Gesamtergebnisses entsprechen. Diese Werte entsprechen der Normierung auf Klein- und Kleinstunternehmen, demzufolge 98% der österreichischen Unternehmen.

Auf Basis der Tabelle 2: Prozentuelle Verteilung der Ergebnisse nach Beschäftigtengrößenklassen wird daher ein Verteilungsschlüssel wie folgt angewendet.

Tabelle 3: Ergebnis der Frage F1. in der Unternehmenskategorie UK1 mit Korrektur für Angleichung zur Realunternehmensverteilung

F1.	Unternehmenskategorie - UK1		
	Anzahl	**Korrektur**	**%**
Ja	29	32,2082163	37,6623377
Nein	48	53,3101511	62,3376623
Gesamt	77	**85,5183673**	100

Tabelle 4: Ergebnis der Frage F1. in der Unternehmenskategorie UK2 mit Korrektur für Angleichung zur Realunternehmensverteilung

F1.	Unternehmenskategorie - UK2		
	Anzahl	**Korrektur**	**%**
Ja	18	9,92996778	94,7368421
Nein	1	0,55166488	5,26315789
Gesamt	19	**10,4816327**	100

Tabelle 5: Ergebnis der Frage F1. mit Korrektur für Angleichung zur Realunternehmensverteilung

F1.	Summe (UK1 + UK2)		
	Anzahl	**Korrektur**	**%**
Ja	47	42,1381841	**48,9583333**
Nein	49	53,8618159	**51,0416667**
Gesamt	96	**96**	100

1.3.2.2 *Forschungsfragen und –antworten*

Hinsichtlich der durchgeführten Umfrage kann nun auf die anfänglich definierte Forschungsfrage eingegangen werden.

„Wie viel Prozent der Klein- und Kleinstunternehmen verwenden eine Softwarelösung zu Erstellung und Verwaltung Ihrer Rechnungen?“

Mit Hilfe der Fragen F1 und F2 konnte ermittelt werden, dass 48,96% der Klein- und Kleinstunternehmer bereits eine Softwarelösung zu Erstellung und Verwaltung Ihrer Rechnungen verwenden.

Zusätzlich konnte gezeigt werden, dass es einen erheblichen Unterschied zwischen Kleinunternehmen und Kleinstunternehmen gibt. So haben 94,74% der Kleinunternehmen eine Softwarelösung zu Erstellung und Verwaltung Ihrer Rechnungen, hingegen nur 37,66% der Kleinstunternehmen eine derartige Software in ihrem Unternehmen.

1.3.3 Zusammenfassung

Die oben durchgeführte Empirie zeigt nun, welches Marktpotential ERP(Enterprise-Resource-Planning)- sowie Verrechnungsmanagementsysteme im Bereich der Klein- und Kleinstunternehmen haben. Anhand der Zahlen dieser Empirie ergibt sich ein potenzielles Marktpotential von approximierten 165.000 Unternehmen (stand 2013), die ihre Geschäftsprozesse hinsichtlich der Verrechnung durch eine angepasste Softwarelösung optimieren könnten.

2. Tarifproblematik

Da sich diese Arbeit primär mit Verrechnungsmanagementsoftwaresystemen beschäftigt, trifft man im Bereich des Verrechnungsmanagements unumgänglich auf Tarife.

Tarife sind grundsätzlich eine preisliche Reglementierung von Dienstleistungen und bilden daher die Basis von Dienstleistungsverrechnungen. Die Bezeichnung „Tarif“ findet man nur im Bereich der Leistungserbringung und wird als solches nicht im Warenhandel verwendet[15]. Dennoch findet man im Handel auch sehr oft reglementierte Rabattierung von Produkten, die anhand von Preislisten definiert werden. Angesichts dessen stößt man auch hierbei auf das gleiche Kernproblem, wie es Tarife haben. Anhand dieser Gegebenheit wird in diesem Kapitel das gemeinsame Kernproblem vereinfacht unter dem Überbegriff von Tarifen behandelt.

Grundsätzlich geht es bei dieser Kernproblematik darum, dass der zu verrechnende Preis von Leistungen bzw. Produkten von vielen Faktoren abhängig ist und daher nicht immer leicht ermittelbar ist. Dieses Problem impliziert bei einer nicht automatisierten Verrechnung und einer steigenden Komplexität der Tarife eine einhergehende Steigerung der personellen Bindung. Eine derartige Bindung wird sehr oft durch das Integrieren von Softwarelösungen in das Verrechnungsmanagement aufgebrochen. Die eingesetzten Softwarelösungen sind meist Geschäftssparten spezifisch und daher schwer universell einsetzbar[16].

Im Zuge dieser Arbeit wird auf solche Softwarelösungen eingegangen und analysiert, wie ein Softwareframework aussehen sollte, das universell einsatzbar ist und folglich auch als Grundlage von spezialisierten Softwarelösungen dienen könnte. Des Weiteren sollte dieses Framework auch als „best practice“ Konzept gesehen werden und als Vorlage für andere Softwarelösungen im Bereich des Verrechnungsmanagements dienen.

---

[15] Zitat DUDEN – „festgesetzter Preis; Entgelt, Gebühr für etwas (z. B. für die Inanspruchnahme von Dienstleistungen)“

[16] ERP Software Liste - 04.05.16 - en.wikipedia.org/wiki/List_of_ERP_software_packages

In diesem Kapitel wird demzufolge eine grundsätzliche Analyse von Tarifen durchgeführt und diskutiert, wie eine Abstraktion von Tarifen aussehen müsste. Aufbauend auf diese Analyse wird ermittelt, wie solche Tarife durch Software umgesetzt werden können und welche Probleme im Zuge einer Implementierung auftreten können.

Demgemäß wird begonnen anhand eines gerichteten Graphen eine Abstraktion von Tarifen zu erreichen. Aufbauend auf diese Abstraktion wird in diesem Kapitel auch definiert, aus welchen Komponenten ein Tarifmodell bestehen sollte. Anschließend werden auf Recommender Systeme für Tarife eingegangen und beschrieben, welche Konzepte hierfür angewendet werden können. Abschließend wird noch beschrieben, wie das definierte Modell anhand der Programmiersprache Java umgesetzt werden kann und auf welche Probleme man bei dieser Umsetzung trifft.

2.1 Tarifanalyse

Im Zuge dieser Arbeit wurden 53 Tarife verschiedener Versicherungen und Dienstleistungsanbieter hinsichtlich ihrer Zusammenstellung und deren Aussagekraft analysiert und als Grundlage für das im folgenden Kapitel definierte Modell verwendet.

2.1.1 Tarifmodell

Mit Hilfe der eingangs erwähnten Tarifanalysen, konnte folgendes Modell erarbeitet werden. Betrachtet man einen Tarif, so findet man vier Komponenten, die sich bei allen Tarifen wiederfinden.

- Tarifeingaben
- Tarifausgaben
- Tarifkonfiguration
- Tariflogik

Des Weiteren kann das Verhalten dieser Komponenten mit folgender Aussage prosaisch beschrieben werden.

„Die Tariflogik ermittelt anhand einer konkreten Tarifkonfiguration und mit Hilfe der Tarifeingaben eine resultierende Tarifausgabe.“

Anhand dieser vier Komponenten und der oben angeführten Aussage wurde ein Modell (siehe Abbildung 1) definiert, welches die Eigenschaften von Tarifen bestmöglich abstrahiert. Bei diesem Modell handelt es sich um einen gerichteten Graphen, da es durchaus vorkommen kann, dass Tarifausgaben wiederum als Eingabe von anderen Tarifen verwendet werden können.

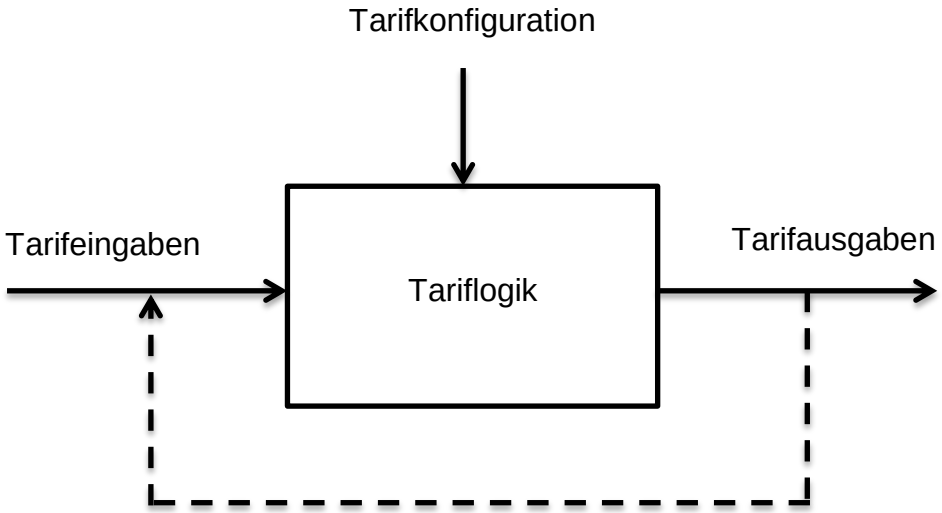


Abbildung 1. Tarifmodell

Formal kann dieses Modell auch wie folgt beschrieben werden.

Man definiere E als Menge aller möglichen Eingabevektoren, A als die Menge aller möglichen Ausgabevektoren und K als Menge aller möglichen Konfigurationsvektoren eines Tarifes. Wenn nun alle Vektoren dieser Mengen als n-Tupel reeller Zahlen definiert sind, so können wir die Tariflogik des Modells wie folgt definieren.

Die Tariflogik sei nun eine Funktion t für die gilt

$$t_{\vec{k}}(\vec{e}) = \vec{a}, \qquad \vec{k} \in K, \vec{e} \in E, \vec{a} \in A$$

Angesichts der Funktionseigenschaften können hier keine weiteren generellen Aussagen getroffen werden, da sich diese Eigenschaften je nach Tariflogik verändern können. Eine Auflistung der Funktionseigenschaften anhand einer Differenzierung der Tariflogik wird im nächsten Kapitel durchgeführt.

2.1.2 Tariflogik

Bei der Tariflogik handelt es sich um den Kern eines jeden Tarifes. Anhand der Tariflogik werden Elemente der Menge E mit Hilfe von Konfigurationselementen der Menge K auf Ausgabeelemente der Menge A abgebildet.

2.1.2.1 *Bijektivität oder Surjektivität*

Wie bereits im vorigen Kapitel erwähnt, kann man die Tariflogik als Funktion definieren. Hinsichtlich der Funktionseigenschaften gibt es aber keine eindeutigen Aussagen. So kann eine Funktion $t_{\vec{k}}$ die eine Tariflogik beschreibt sowohl surjektiv als auch in manchen Fällen bijektiv sein.

Würden wir ein Skalar als Eingabevektor $\vec{e}$, sowie ein Skalar als Konfigurationsvektor $\vec{k}$ in die Funktion $t_{\vec{k}}$ einsetzen,

$$\vec{e} = (e_1), \qquad e_1 \dots gefahrene\ Strecke$$

$$\vec{k} = (k_1), \qquad k_1 \dots Preis\ gefahrene\ Strecke$$

$$t_{\vec{k}}(\vec{e}) = \vec{k} * \vec{e}$$

so wäre die Funktion $t_{\vec{k}}$ eindeutig bijektiv und demgemäß würde gelten, dass

$$t_{\vec{k}}^{-1}\left(t_{\vec{k}}(\vec{e})\right) = \vec{e}, \qquad \vec{k} \in K, \vec{e} \in E$$

Wäre die Funktion hingegen wie folgt definiert, wäre sie eindeutig surjektiv.

$$t_{\vec{k}}(\vec{e}) = \begin{cases} k_1 * e_1, & falls\ k_1 * e_1 > k_2 * e_2 \\ k_2 * e_2, & falls\ k_1 * e_1 \leq k_2 * e_2 \end{cases}$$

Folglich würde gelten

$$t_{\vec{k}}^{-1}\left(t_{\vec{k}}(\vec{e})\right) \neq \vec{e}\,, \qquad \vec{k} \in K\,, \vec{e} \in E$$

2.1.2.2 *Komposition von Tarifen*

Da Tarifausgaben auch Tarifeingaben sein können, müssen Tariflogikfunktionen auch verkettet werden können. Im Falle des abstrakten Graphen-Modells aus dem vorhergehenden Kapitel ist dies durch eine Rückführung des Tarifausganges auf den Tarifeingang beschrieben. Formal definiert bedeutet das, dass

$$(t_{\overrightarrow{k_1}}^{(1)} \circ t_{\overrightarrow{k_2}}^{(2)})(\vec{e}) = t_{\overrightarrow{k_2}}^{(2)}(t_{\overrightarrow{k_1}}^{(1)}(\vec{e}))\,, \qquad \overrightarrow{k_1} \in K_1,\ \overrightarrow{k_2} \in K_2\,, \vec{e} \in E$$

Eine Verkettung dieser Funktionen ist nur selten direkt möglich, da Eingabevektoren von Tariflogikfunktionen sehr variieren können. In solchen Fällen ist eine Mappingfunktion m nötig, um eine Komposition von mehreren Tarife erstellen zu können.

$$(t_{\overrightarrow{k_1}}^{(1)} \circ t_{\overrightarrow{k_2}}^{(2)})(\vec{e}) = t_{\overrightarrow{k_2}}^{(2)}(m_{t^{(1)}t^{(2)}}(t_{\overrightarrow{k_1}}^{(1)}(\vec{e})))\,, \qquad \overrightarrow{k_1} \in K_1,\ \overrightarrow{k_2} \in K_2\,, \vec{e} \in E$$

Für beide Varianten der Verkettung ist es wichtig, dass die möglichen Ausgabevektoren der Funktion $t^{(1)}$ bzw. $m(t^{(1)})$ eine Teilmenge der möglichen Eingabevektoren der Funktion $t^{(2)}$ ist. Formal entspricht dies

$$A_{t^{(1)}} \subseteq E_{t^{(2)}}$$

In den vorangegangenen Kapiteln sollten nun die wichtigsten Eigenschaften der besagten Tariflogikfunktion beschrieben worden sein. Weitere Eigenschaften dieser Funktion sind abhängig von den verwendeten mathematischen Komponenten, welche als Funktion $t_{\vec{k}}$ zusammengefasst sind. Sie sollten aber hinsichtlich der oben genannten Eigenschaften beschränkt sein.

2.1.3 Tariftransformation

Sehr viele Tarife werden prosaisch manifestiert. Eine Transformation zu einer mathematischen Komposition kann in der Regel sehr komplex sein. Dennoch

hat diese Transformation sowohl eine theoretische als auch praktische Relevanz. In der Theorie hilft diese Transformation die mathematische Diversität von Tarifen zu analysieren. In der Praxis hingegen wird eine Transformation benötigt, um den Programmcode für eine Softwarelösung zu modellieren(siehe Kapitel 2.3).

Die oben erwähnte Transformation sollte in beiden genannten Anwendungsbereichen unter Beachtung der Trennung von Tariflogik und Tarifkonfiguration durchgeführt werden. Im konkreten bedeutet das, dass man primär eine Trennung von möglichen variablen Elementen (Tarifkonfiguration) und statischen Elementen (Tariflogik) anstreben sollte. Anhand des folgenden Beispiels sollte eine solche Transformation (von einem prosaisch hin zu einem mathematisch definierten Tarif) beschrieben werden.

Für dieses Beispiel wird der Tarif eines Berge- und Abschleppunternehmens herangezogen. Die Komplexität dieses Tarifes ist aus demonstrativen Zwecken sehr beschränkt, dennoch sollte anhand der anschließenden Analyse die erwähnte Trennung von Tariflogik und Tarifkonfiguration dargestellt werden.

Die unten angeführte Tabelle (siehe Tabelle 6) ist die Basis der Taxierung eines Bergeverfahrens.

Tabelle 6: Tarifauszug eines Berge- und Abschleppunternehmen[17]

Bergefahrzeug mit Kran (Mindestverrechnung 1 Stunde)	€ 100,00
Bergungsfachkraft (Mindestverrechnung 1 Stunde)	€ 85,00
Bergehelfer (Mindestverrechnung 1 Stunde)	€ 42,00
Anschlagmittel (Gurte, Ketten etc.) / Stück	€ 6,00
Reinigung Bergefahrzeug	€ 42,00

Analysiert man diese Tabelle, so findet man Verrechnungsposten mit zugewiesenen Wertigkeiten und dessen Beschränkungen bzw. Reglementierungen. Der grau hinterlegte Text kann als variabel angenommen werden und sollten daher

[17] http://www.abschleppen-bergen.at/de/preise , Abschleppdienst MACHALEK - Anhang 6

als Konfigurationsvektor $\vec{k}$ transformiert und definiert werden. Im Eingabevektor $\vec{e}$ sollten im konkreten Beispiel alle nötigen Werte übergeben werden, die eine Taxierung der Positionen zulassen. Durch diese Beschränkungen kann nun eine mathematische Tariflogik entwickelt werden.

Der Eingabe-, Konfigurations- und Ausgabevektor ist daher wie folgt zu definieren:

$$\vec{e} = \begin{pmatrix} e_1 \\ \vdots \\ e_8 \end{pmatrix}, \qquad e_1, e_2, e_3, e_7 \in \mathbb{N}, \qquad e_4, e_5, e_6 \in \mathbb{R}^+, \qquad e_8 \in \{0,1\}$$

$$e_1 \dots \textit{Anzahl Bergefahrzeuge}$$

$$e_2 \dots \textit{Anzahl Bergungsfachkräfte}$$

$$e_3 \dots \textit{Anzahl Bergehelfer}$$

$$e_4 \dots \textit{Kummulierte Stunden für Bergefahrzeuge}$$

$$e_5 \dots \textit{Kummulierte Stunden für Bergungsfachkräfte}$$

$$e_6 \dots \textit{Kummulierte Stunden für Bergehelfer}$$

$$e_7 \dots \textit{Stückzahl Anschlagmittel}$$

$$e_8 \dots \textit{Reinigung Bergefahrzeuge } (Ja/Nein)$$

$$\vec{k} = \begin{pmatrix} k_1 \\ \vdots \\ k_8 \end{pmatrix}, \qquad k_1, \dots, k_8 \in \mathbb{R}^+$$

$$k_1, \dots, k_5 \dots \textit{Preise von Verrechnungsposten } 1-5$$

$$k_6, \dots, k_8 \dots \textit{Mindestverrechnung von Posten } 1-3$$

$$\vec{a} = (a), \qquad a \in \mathbb{R}^+$$

$$a \dots \textit{Resultierender Preis}$$

Die Tariflogikfunktion $t_{\vec{k}}$ erschließt sich dann wie folgt.

$$t_{\vec{k}}(\vec{e}) = \sum_{n=1}^{3} \left((e_n * k_n) + max(e_{n+3} - (e_n * k_{n+5}), 0) * k_n \right) + \sum_{n=7}^{8} (e_n * k_n)$$

Betrachtet man nun diese Funktion, so findet man hier eine gut erkennbare Trennung zwischen Logik und Konfiguration. Diese Tariffunktion kann nun auch als Ausgangpunkt für eine weitere Transformation hin zu einem Programmcode verwendet werden. Eine passende Programmarchitektur für das Integrieren einer in Programmcode transformierten Tariffunktion wird im Kapitel 2.3 beschrieben.

2.2 Recommender Systems für Tarife

In diesem Kapitel wird nun analysiert, wie Tarifempfehlungen auf Basis des oben definierten Tarifmodells getroffen werden und welche grundsätzlichen Probleme hierbei auftreten können. Tarifempfehlungen sind durchaus ein wichtiges Themenfeld, da gleiche Produkte bzw. Dienstleistungen in der Regel auch mit unterschiedlichen Tarifen taxiert werden können. Aus Sicht des wirtschaftlichen Interesses ist es in solchen Fällen natürlich notwendig, eine optimale Tarifwahl zu treffen.

Die nun durchgeführten Analysen sollten grundsätzlich dabei helfen, ein praxisnahes und adäquates Recommendersystem zu entwickeln. Dieses wiederum sollte auch ein Teil des bereits angesprochenen konzeptionellen Programmframeworks sein. Des Weiteren sollten die ermittelten Einschränkungen auch dazu beitragen, die Grenzen bei einer softwarebasierten Umsetzung greifbarer zu machen. Hierfür sollten daher in erster Line die wichtigsten Anwendungsfälle (Use Cases) einer Tarifempfehlung definiert werden.

Grundsätzlich gibt es zwei Kernanwendungsfälle – UC1 und UC2. Diese unterscheiden sich hauptsächlich durch das Vorhandensein einer diskreten Tarifeingabe.

UC1. Eine Tarifempfehlung wird anhand einer bereits vorhandenen vollständigen Tarifeingabe getroffen. Im Zuge dieses Use Cases wird ermittelt, mit welcher Kombination aus Tariflogik und Tarifkonfiguration ein optimales Ergebnis erzeugt werden kann.

UC2. Dieser Use Case ist hinsichtlich der Voraussetzungen ähnlich dem UC1. Der Unterschied findet sich aber in der Vollständigkeit der Tarifeingabe.

Bei diesem Anwendungsfall befinden sich nämlich in der Tarifeingabe Lücken. Neben dem Optimierungskriteriums vom UC1 müssen nun auch diese Lücken als weitere Parameter in das Optimierungskriterium eingebunden werden.

Hinsichtlich dieser Unterteilung ist es sinnvoll, jeden dieser Anwendungsfälle getrennt zu analysieren und die auftretenden Kernprobleme zu definieren. Generell können beide Anwendungsfälle auf ein mathematisches Problem reduziert werden, welches aus dem Bereich der Optimierungsverfahren stammt. Ad hoc ist es auch möglich, die in den Anwendungsfällen auftretenden Optimierungsprobleme auch als Vektoroptimierungsprobleme zu kategorisierten. Im Verlauf der nächsten Kapitel wird auf diese Kategorisierung eingegangen und beschrieben, warum diese Einordnung möglich ist.

2.2.1 Was ist eigentlich eine Tarifempfehlung

Bevor die angesprochenen Anwendungsfälle beleuchtet werden, sollte zunächst definiert werden, was man unter einer Tarifempfehlung versteht. Eine Tarifempfehlung könnte wie folgt prosaisch beschrieben werden.

„Eine Tarifempfehlung ist ein 2-Tuble (Tariftupel) aus Tariflogik und Tarifkonfiguration, das bei einer gegebenen Tarifeingabe die optimale Tarifausgabe hinsichtlich additiver optionaler Randbedingungen erzeugt“

In dieser Beschreibung wird unter anderem von einem Optimum gesprochen. Allgemein kann man in der Mathematik bei einem Optimum[18] von einem Extremum sprechen, welches entweder ein Minimum oder Maximum ist.

Zum besseren Verständnis sollte auch eine formale Definition entwickelt werden. Das oben beschriebene Verhalten bei einer Tarifempfehlung kann daher mathematisch formal wie folgt definiert werden.

[18] Zitat DUDEN – „(unter den gegebenen Voraussetzungen, im Hinblick auf ein Ziel) höchstes erreichbares Maß, höchster erreichbarer Wert“

Sei $n \in \mathbb{N}$ die Anzahl der Tariflogiken und $\overrightarrow{k_{(y)}^{(x)}}$ eine zu $t^{(x)}$ gehörige Tarifkonfiguration aus der Menge der möglichen Tarifkonfigurationen $K^{(x)}$, wobei gilt, dass $x, y \in \mathbb{N}$ ist. Die Menge aller möglichen Tariftupel T kann daher wie folgt definiert werden.

$$T = \left\{ \left(t^{(x)}, \overrightarrow{k_{(y)}^{(x)}} \right) \middle| \left(x \geq 1 \wedge x \leq n \right) \wedge \left(y \geq 1 \wedge y \leq \left| K^{(x)} \right| \right) \wedge \overrightarrow{k_{(y)}^{(x)}} \in K^{(x)} \right\}$$

Die optimale Tariflogikfunktion $t^*_{\overrightarrow{k^*}}$ mit einem Eingabevektor $\vec{e}$ aus der Menge E ist daher wie folgt definiert.

$$t^*_{\overrightarrow{k^*}}(\vec{e}) = \min_{t \in T} g(t,\vec{e})$$

$$\vec{g}: (T, E) \rightarrow \mathbb{R}^d \qquad g(t,\vec{e}) = f_{t_2}(\vec{e}) \; : \; f \equiv t_1 \wedge d \in \mathbb{N} \wedge \vec{e} \in \mathbb{R}^d$$

Bisher wurde nur formal definiert, was ein optimales Tariftupel ist, dennoch stellt sich anschließend aber die Frage, wie ein optimales Tariftupel ermittelt werden kann. Betrachtet man diese formale Definition, so kann man sehr gut erkennen, dass ein anwendbares Optimierungskriterium auf Basis von n-Tupel reeller Zahlen arbeiten sollte. Da sich solche Optimierungskriterien mit Vektoren beschäftigen, kann das oben definierte Optimierungsproblem als Vektoroptimierungsproblem gesehen werden. Man spricht hier auch von einem multiobjektiven Optimierungsproblem.

Grundsätzlich kann man bei einer Vektoroptimierung nicht ohne weiteres sagen, welches Ergebnis optimal ist. Dennoch kann man zum Beispiel durch das Einschränken der Ergebnismenge das Optimum konkretisieren. Diese Einschränkung kann mitunter durch das Betrachten der Pareto-Fronten durchgeführt werden[**3**].

Forschungsarbeiten im Zusammenhang mit Vektoroptimierungsverfahren liefern stetig neue Methoden, um multiobjektive Optimierungsprobleme bestmöglich lösen zu können. Viele dieser Verfahren arbeiten mit der bereits erwähnten Pareto-Front. Andere Verfahren wiederum verwenden eine Transformation der Ergebnisvektoren hin zu Ergebnisskalaren. Dennoch benötigen viele dieser Verfahren weitere Informationen um ein Optimum zu bestimmen[**4**].

Genau aus diesem Grund wurden auch in der eingangs angeführten prosaischen Definition die optionalen Randbedingungen eingeführt, die dem Optimierungsverfahren beschreiben sollten, was als Optimum gesucht wird. Bei diesen Randbedingungen kann es sich zum Beispiel um Gewichtungsverteilungen, aber auch um Grenzwertschranken handeln. Dennoch können manche Optimierungsverfahren aber auch ohne weitere Randbedingungen die gesuchten Optima bestimmen. Hierbei würde man von einem vorzugslosen Optimierungsverfahren sprechen **[5].** Im Generellen betrachtet, überwiegen aber Optimierungsverfahren mit zu definierenden Randbedingungen.

Anhand des Eingangs formal definierten Optimierungsproblems, kann man sehr gut erkennen, dass eine höhere Komplexität der Optimierung sowohl durch Zunahme der Vektorendimensionen, als auch durch eine Steigerung der Tariftupel bewirkt werden kann. Hinsichtlich dieser Beobachtung wird im Zuge der Ausarbeitung des UC1 auch ein Fokus auf diese Problematiken gerichtet.

2.2.2 UC1 – Die Tarifempfehlung als Vektorenoptimierungsproblem

Wie bereits erwähnt, kann das Empfehlen eines Tarifes, wie es im UC1 auftritt, als mathematisches Problem gesehen werden und entspricht der Suche des Optimums in einer Ergebnismenge. Als Ergebnismenge R kann man die Menge aller generierbaren Tarifausgaben definieren, die man mit Hilfe aller möglichen Tariftupel bei einer konkreten Tarifeingabe erzeugen kann.

Zu der bereits getroffenen Kategorisierung als Vektoroptimierungsproblem kann auch die Einordnung in die diskrete Mathematik gemacht werden, und begründet sich durch das praktische Verhalten des oben beschriebenen Modells. So gibt es bei einer konkreten Tarifeingabe und einer endlichen Anzahl von Tariflogiken mit einer endlichen Anzahl von dazugehörigen Tarifkonfigurationen auch nur eine endliche Ergebnismenge. Die Beschränkung auf eine konkrete Eingabe ist in diesem Kontext sehr wichtig, da eine Optimierung hinsichtlich Tarifkonfigurationen und Tariflogik durchgeführt wird und nicht gemäß der Eingabewerte. Diese Art der Optimierung ist Teil des nächsten Kapitels und benötigt eine separate Behandlung.

Das asymptotische Verhalten des oben beschriebenen mathematischen Problems ist daher abhängig von der Anzahl der möglichen Tariflogiken, Tarifkonfigurationen und Tarifkompositionen (siehe Kapitel 2.1.2.2). Um eine allgemeine Aussage hinsichtlich der Problemkomplexität treffen zu können, sollten die Ergebnismengen von Basisfällen analysiert werden und das asymptotische Verhalten deren Kardinalitäten bestimmt werden. Grundsätzlich kann man fünf Basisfälle definieren.

$Fall\ 1$: Sei $n \in \mathbb{N}$ die Anzahl der möglichen Tariflogiken und $t^{(x)}$ eine konkrete Tariflogik mit einer dazugehörigen Tarifkonfiguration $\overrightarrow{k^{(x)}}$ aus der Menge der möglichen Tarifkonfigurationen$K^{(x)}$, wobei gilt, dass $x \in \mathbb{N}$ und $x \geq 1 \wedge x \leq n$ ist, dann ergibt sich daraus die Ergebnismenge R^1 ,die wie folgt definiert werden kann.

$$R^1 = \left\{ t^{(x)}_{\overrightarrow{k^{(x)}}}(\vec{e}) \middle| x \geq 1 \wedge x \leq n \wedge \overrightarrow{k^{(x)}} \in K^{(x)} \wedge \left|K^{(x)}\right| = 1 \right\}$$

Das Verhalten der Kardinalität von R^1 wird nun durch die Funktion r_1 abgebildet.

$$r_1 \colon \mathbb{N} \to \mathbb{N}$$

$$|R^1| \equiv r_1(n) = n$$

$$\boldsymbol{\mathcal{O}\left(r_1(n)\right) = n}$$

$Fall\ 2$: Sei $\overrightarrow{k_x}$ eine von n Tarifkonfigurationen der Tariflogik t aus der Menge der möglichen Tarifkonfigurationen K, wobei gilt, dass $x, n \in \mathbb{N}$ sind und $x \geq 1 \wedge x \leq n$ ist, dann ist die Ergebnismenge R^2 wie folgt definiert.

$$R^2 = \left\{ t_{\overrightarrow{k_x}}(\vec{e}) \middle| x \geq 1 \wedge x \leq n \wedge \overrightarrow{k_x} \in K \wedge |K| = n \right\}$$

Das Verhalten der Kardinalität von R^2 wird nun durch die Funktion r_2 abgebildet.

$$r_2 \colon \mathbb{N} \to \mathbb{N}$$

$$|R^2| \equiv r_2(n) = n$$

$$\boldsymbol{\mathcal{O}\big(r_2(n)\big) = n}$$

$Fall\ 3$: Sei $\overrightarrow{k_y^{(x)}}$ eine von $m^{(x)}$ Tarifkonfigurationen der Tariflogik $t^{(x)}$ aus der Menge der möglichen Tarifkonfigurationen $K^{(x)}$, wobei gilt, dass $x, y, n \in \mathbb{N}$ sind und $m^{(x)} \in \mathbb{N}$ ist, aber auch, dass $x \geq 1 \wedge x \leq n$ sowie $y \geq 1 \wedge y \leq m^{(x)}$ ist, dann ist die Ergebnismenge R^3 wie folgt definiert

$$R^3 = \left\{ t^{(x)}_{\overrightarrow{k_y^{(x)}}}(\vec{e}) \;\middle|\; \begin{array}{c} x \geq 1 \wedge x \leq n \\ y \geq 1 \wedge y \leq m^{(x)} \\ \overrightarrow{k_y^{(x)}} \in K^{(x)} \\ \left|K^{(x)}\right| = m^{(x)} \\ \left|\bigcup_{i=1}^{n} K^{(i)}\right| = \sum_{i=1}^{n} \left|K^{(i)}\right| \end{array} \right\}$$

Das Verhalten der Kardinalität von R^3 wird nun durch die Funktion r_3 abgebildet.

$$Sei\ M = \left\{m^{(1)}, \dots, m^{(n)}\right\}\ dann\ ist$$

$$r_3 \colon \mathbb{N}^n \to \mathbb{N}$$

$$|R^3| \equiv r_3(M) = \sum_{i=1}^{|M|} m^{(i)}$$

$$\boldsymbol{\mathcal{O}\big(r_3(M)\big) = |M| * \sup M}$$

In den ersten drei Fällen wurde anhand von variablen Tariflogiken und Tarifkonfigurationen das asymptotische Verhalten der Ergebnismengen bestimmt. In den weiteren zwei Fällen kommt nun die Variabilität bei der Anwendung von Tarifkompositionen hinzu.

Prinzipiell kann die Komposition von Tarifen als Anwendungsgebiet der abzählenden Kombinatorik gewertet werden. Demgemäß müssen die definierten Tariflogiken als unterscheidbare Objekte gesehen werden. Auch muss man hinsichtlich der im Kapitel 2.1.2.2 beschriebenen Tarifkompositionen von einer Variation sprechen. Die Gründe dafür liegen in der Tatsache, dass es bei einer veränderten Reihenfolge von kombinierten Tarifen auch zu Änderungen im resultierenden Ergebnis kommen kann.

Die Einteilung der nun besprochenen Fälle basiert auf Grundlage der abzählenden Kombinatorik und kategorisiert grundsätzlich anhand der wiederholten Verwendung von Tariflogiken bei einer Tarifkomposition. Hierbei würde man von einer Variation mit Wiederholung bzw. einer Variation ohne Wiederholung sprechen.

$Fall\ 4$ ($Variation\ ohne$ Wiederholung): Sei $n \in \mathbb{N}$ die Anzahl der möglichen Tariflogiken und $\overrightarrow{k^{(x)}}$ eine zu $t^{(x)}$ gehörige Tarifkonfiguration aus der Menge der möglichen Tarifkonfigurationen $K^{(x)}$, wobei gilt, dass $x \in \mathbb{N}$ ist. Die Menge aller Tariflogiken T kann nun wie folgt definiert werden.

$$T = \left\{ t^{(x)}_{\overrightarrow{k^{(x)}}} \middle|\ x \geq 1 \wedge x \leq n \wedge \overrightarrow{k^{(x)}} \in K^{(x)} \wedge \left|K^{(x)}\right| = 1 \right\}$$

Des Weiteren solle gelten, dass bei einem $l \in \mathbb{N}$ für jedes $l \geq 1 \wedge l \leq n$ eine Menge P^l existiert, welche die Folgen aller *k-Permutationen ohne Wiederholung* mit der Länge l aus der Menge T beinhaltet.

Demgemäß ist die Ergebnismenge R^4 wie folgt definiert.

$$R^4 = \bigcup_{z=1}^{n} \left(\bigcup_{y=1}^{z} p_{(1)} \circ \ldots \circ p_{(y)}(\vec{e}),\ p \in P^z \right)$$

Das Verhalten der Kardinalität von R^4 wird nun durch die Funktion r_4 abgebildet.

$$r_4 \colon \mathbb{N} \to \mathbb{N}$$

$$|R^4| \equiv r_4(n) = \sum_{a=1}^{n} \binom{n}{a} * a = \sum_{a=1}^{n} \frac{n!}{(n-a)!} = n! * \sum_{a=1}^{n} \frac{1}{(n-a)!}$$

$$\boldsymbol{\mathcal{O}\left(r_4(n)\right) = n!}$$

$Fall\ 5$ ($Variation\ mit$ Wiederholung): Sei $n \in \mathbb{N}$ die Anzahl der möglichen Tariflogiken, $m \in \mathbb{N}$ die maximale Länge einer Tarifkomposition und $\overrightarrow{k^{(x)}}$ eine zu $t^{(x)}$ gehörigen Tarifkonfiguration aus der Menge der möglichen Tarifkonfigurationen $K^{(x)}$, wobei gilt, dass $x \in \mathbb{N}$ ist, dann kann die Menge aller Tariflogiken T wie folgt definiert werden.

$$T = \left\{ t^{(x)}_{\overrightarrow{k^{(x)}}} \middle| \, x \geq 1 \wedge x \leq n \wedge \overrightarrow{k^{(x)}} \in K^{(x)} \wedge \left|K^{(x)}\right| = 1 \right\}$$

Des Weiteren soll gelten, dass bei einem $l \in \mathbb{N}$ für jedes $l \geq 1 \wedge l \leq m$ eine Menge P^l existiert, welche die Folgen aller *Permutationen mit Wiederholung* der Länge l aus der Menge T beinhaltet.

Demgemäß ist die Ergebnismenge R^5 wie folgt definiert.

$$R^5 = \bigcup_{z=1}^{m} \bigcup_{y=1}^{z} p_{(1)} \circ \ldots \circ p_{(y)}\,(\vec{e}), \;\; p \in P^z$$

Das Verhalten der Kardinalität von R^5 wird nun durch die Funktion r_5 abgebildet.

$$r_5 \colon \mathbb{N}^2 \to \mathbb{N}$$

$$|R^5| \equiv r_5(n,m) = \sum_{a=1}^{m} n^a$$

$$\boldsymbol{\mathcal{O}\left(r_5(n,m)\right) = n^m}$$

In der Praxis wird man vermehrt auf Kombinationen dieser fünf Fälle treffen. Dennoch kann man anhand der oben beschriebenen Basisfällen feststellen, dass es sich im Falle der Komplexität des UC1 um ein Problem handelt, welches eine starke Korrelation zur Anzahl der eingesetzten Tarifen und Tarifkonfigurationen aufweist. Zudem kann man auch sehr gut erkennen, dass durch die Möglichkeit der Tarifkompositionen das asymptotische Verhalten der Ergebnismenge eine markante Steigerung erfährt.

Angesichts der durchgeführten Analysen ist der Umfang der möglichen optimalen Lösungen ausreichend bestimmt, dennoch stellt sich die Frage, wie nun ein solches Optimum aus dieser Ergebnismenge bestimmt werden kann. Hierzu hilft die zuvor getroffene Zuordnung zu den Vektoroptimierungsproblemen. Wie bereits mehrfach erwähnt wurde, sind alle Tarifausgaben, welche als Ergebnismenge zusammengefasst sind, n-Tupel reeller Zahlen und somit Vektoren. Somit ist es in der Regel nicht möglich, nur anhand der Tarifausgabe das Optimum zu bestimmen. Hierfür wird noch einen Verfahren benötigt, welches als Mindestmaß die Menge der in Frage kommenden optimalen Lösungen minimieren kann und im besten Falle ein eindeutiges Optimum bestimmen kann. Infrage kommende Verfahren werden zusammenfassend als Vektoroptimierungsverfahren bezeichnet.

Allgemein können diese Vektoroptimierungsverfahren auf vier unterschiedliche Konzepte aufbauen. Einerseits gibt es das vorzugslose Konzept, welches ohne jegliche Randbedingungen eine optimale Lösung finden kann und anderseits gibt es Verfahren, die mit Hilfe von Randbedingung Lösungen ermittelten können. Letztere Verfahren wiederum basieren auf Apriori-, Posteriori- und Interaktivmethodiken. Grundsätzlich unterscheiden sich die Methoden mit Randbedingungen nur hinsichtlich des Zeitpunktes, bei dem eine Randbedingung definiert bzw. angepasst wird **[5].**

Prinzipiell können alle Verfahren (mit und ohne Randbedingungen) angewendet werden, um eine Tarifempfehlung zu treffen. Dennoch kann es bei steigender Komplexität sinnvoller sein, mit Randbedingungen zu arbeiten. In der Regel sollte dies auch kein Problem sein, da sehr viele Tarifempfehlungen ohnehin interaktiv durchgeführt werden.

Die mathematischen Analysen dieses Anwendungsfalls können nun weitestgehend abgeschlossen werden. Wie eine optimale Integration dieser Optimierungsverfahren in einem Softwaresystem aussieht, wird noch im Kapitel 2.3 näher beleuchtet.

2.2.3 UC2 – Tarifempfehlung mit variablem Eingabevektor

In diesem Kapitel wird nun auf die Problematik eingegangen, dass das Vektoroptimierungsverfahren zur Bestimmung des optimalen Tarifes (siehe Kapitel 2.2.2) keine endliche Ergebnismenge als Grundlage besitzt. Grund für diese Abweichung ist eine Veränderung des Eingabevektors. Wie bereits in der Einleitung definiert, ist bei diesem Anwendungsfall keine konkrete Eingabe vorhanden. Infolge dessen grenzt sich dieser Anwendungsfall auch drastisch vom UC1 ab. Da es im Zuge dieses Anwendungsfalls nur einen teilweise gefüllten Eingabevektor gibt, kann man keine endliche Ergebnismenge erzeugen.

Für eine Analyse dieses Anwendungsfalls empfiehlt sich auch hier, das mathematische Tarifmodel einzubeziehen und schrittweise die Kernprobleme abzuleiten. Betrachten wir nun das mathematische Modell, welches im Kapitel 2.1.1 definiert worden ist.

$$t_{\vec{k}}(\vec{e}) = \vec{a}, \qquad \vec{k} \in K, \vec{e} \in E, \vec{a} \in A$$

In diesem Modell ist die Tariflogik eine mathematische Funktion $t_{\vec{k}}$,welche durch einen Eingabevektor $\vec{e}$ den Ausgabevektor $\vec{a}$ beschreibt. Im UC1 war ein konkreter Eingabevektor vorhanden. Dies hatte zur Folge, dass man durch Bilden aller möglichen Tariftupel einer bestimmten Tarifeingabe eine endliche Ergebnismenge mit Ausgabevektoren ermittelten konnte, welche abschließend nur mehr hinsichtlich des Optimums durchsucht werden musste. Da aber nun im UC2 der Eingabevektor auch variable Vektorelemente besitzt, kann eine einfache Suche nicht mehr angewendet werden. In diesem Fall könnte ein Eingabevektor $\vec{e}_1$ wie folgt aussehen.

$$\vec{e}_1 = \begin{pmatrix} 1{,}5 \\ x_1 \\ x_2 \end{pmatrix}, \qquad x_1, x_2 \in \mathbb{R}$$

Da nun zu Beginn des Tarifempfehlungsvorganges x_1 und x_2 variabel sind, kann es natürlich vorkommen, dass der Ausgabevektor $\vec{a}$ auch Vektorelemente besitzt, die x_1 und x_2 beinhalten. Ein Beispiel anhand einer beliebigen Tariffunktion $t^{(1)}_{\overrightarrow{k_1}}$ mit einer Tarifkonfiguration $\overrightarrow{k_1}$ könnte derart aussehen.

$$t^{(1)}_{\overrightarrow{k_1}}(\vec{e}_1) = \begin{pmatrix} 25 \\ 4x_1 + 3x_2 \\ 8x_1^2 - x_2 \end{pmatrix}$$

Problematisch bei dieser Art von Ausgabevektoren ist, dass durch das Existieren von variablen Elementen nicht jedes Vektoroptimierungsverfahren funktioniert. So ist es zum Beispiel nicht ohne weitere Schritte möglich eine Paretofront zu bilden, da man keine konkreten Werte zur Auswertung besitzt.

Um nun doch eine Vektoroptimierung durchführen zu können, müssen die variablen Elemente bestimmt werden. In den nächsten zwei Kapiteln werden nun zwei Wege definiert, anhand dessen es möglich ist, auf Basis der gegeben Voraussetzungen, ein Optimum zu ermitteln. Die nun besprochenen Möglichkeiten unterscheiden sich grundsätzlich anhand der Notwendigkeit von Nebenbedingungen und der Definitionsgrundlagen des Optimierungskriteriums. Grundsätzlich wird das Optimierungskriterium des ersten Verfahrens auf Basis des Ausgabevektors gebildet, wohingegen das Optimierungskriterium des zweiten Verfahrens auf Basis des Eingabevektors definiert wird. Beim zweiten Verfahren müssen zudem aber zusätzliche Nebenbedingung auf Basis des Ausgabevektors gebildet werden.

2.2.3.1 *Optimierung anhand des Ausgabevektors*

Wie bereits im vorangegangenen Kapitel beschrieben wurde, gibt es mehrere Möglichkeiten um ein multiobjektives Optimierungsproblem mit variablen Elementen zu lösen. Für den konkreten Fall würde sich unter anderem die s.g. Skalarisierung sehr gut anbieten. Grundsätzlich wird im Falle der Skalarisierung ein mehrdimensionales Optimierungsproblem auf ein eindimensionales skalares Problem transformiert **[4]**. Diese Transformation ermöglicht es in weiterer Folge, ein skalares Optimierungsverfahren anzuwenden, um die variablen Eingabeelemente zu bestimmen.

Auf Grundlage des oben beschriebenen Optimierungsproblems (siehe Kapitel 2.2.1) muss in erster Linie eine Anpassung der Mappingfunktion $g(t,\vec{e})$ durchgeführt werden, damit die Skalarisierung integriert werden kann. Um das Optimum ermitteln zu können, wird anhand einer Gewichtung, die Wichtigkeit der einzelnen Dimensionen definiert. Somit muss die Mappingfunktion g durch einen Gewichtungsvektor erweitert werden. Bei genauerer Betrachtung kann man feststellen, dass es sich bei der Anpassung um eine Kapselung der ursprünglichen Mapping Funktion handelt. Formalisieren könnte man das daher wie folgt.

Sei $n \in \mathbb{N}$ die Anzahl der Tariflogiken und $\overrightarrow{k_{(y)}^{(x)}}$ eine zu $t^{(x)}$ gehörige Tarifkonfiguration aus der Menge der möglichen Tarifkonfigurationen$K^{(x)}$, wobei gilt, dass $x, y \in \mathbb{N}$ ist. Die Menge aller möglichen Tariftupel T kann nun wie folgt definiert werden:

$$T = \left\{ \left(t^{(x)}, \overrightarrow{k_{(y)}^{(x)}} \right) \middle| \left(x \geq 1 \wedge x \leq n \right) \wedge \left(y \geq 1 \wedge y \leq \left| K^{(x)} \right| \right) \wedge \overrightarrow{k^{(x)}} \in K^{(x)} \right\}$$

Des weiteren sei $\vec{w} \in \mathbb{R}^d$ der frei bestimmbare Gewichtungsvektor, wobei gilt, dass $d \in \mathbb{N}$ der Vektordimension des Ausgabevektors $\vec{e}$ entspricht und dass

$$\sum_{n=1}^{d} w_n = 1$$

ist, dann kann die Mappingfunktion $g(t,\vec{e},\vec{w})$ wie folgt definiert werden.

$$g\colon (T, E, \mathbb{R}^d) \rightarrow \mathbb{R} \qquad g(t,\vec{e},\vec{w}) = \sum_{n=1}^{d} w_n * a_n(t,\vec{e}) \ : \ d \in \mathbb{N} \wedge \vec{w} \in \mathbb{R}^d$$

$$\vec{a}\colon (T, E) \rightarrow \mathbb{R}^d \qquad a(t,\vec{e}) = f_{t_2}(\vec{e}) \ : \ f \equiv t_1 \wedge d \in \mathbb{N}$$

Da sich durch das Abändern der Mappingfunktion die variablen Terme nicht auflösen, muss bei diesem Verfahren noch ein weiter Schritt durchgeführt werden. Grundsätzlich geht es bei diesem Schritt um die Suche des globalen Maximums bzw. Minimums der neuen Mappingfunktion. Anhand der globalen Extremwerte können die variablen Tarifeingabevektorelemente für ein konkretes

Tariftupel bestimmt werden. Formal kann das nun wie folgt beschrieben werden.

Sei I die Indexmenge eines konkreten Eingabevektors $\vec{e}$, welche beschränkt auf die nicht variablen Vektorelemente ist, dann ist Menge der möglichen Eingabevektoren $E^*_{\vec{e}}$ welche dem konkreten Eingabevektor $\vec{e}$ zugrunde liegen, wie folgt definiert.

$$E^*_{\vec{e}} = \{\vec{x} \mid \vec{x} \in E \ \wedge \ \forall\, i \in I : x_i = e_i \ \} \ \Rightarrow \ E^*_{\vec{e}} \subseteq E : \vec{e} \ \in E$$

Aufbauend kann man nun sagen, dass bei einem Vektor $\overrightarrow{e_{opt}} \in E^*_{\vec{e}}$, die Tariflogikfunktion $t^*_{\overrightarrow{k^*}}$ genau dann optimal ist, wenn folgendes gilt.

$$t^*_{\overrightarrow{k^*}}(\vec{e}) = \min_{t \in T}\left(\min_{\overrightarrow{e_{opt}} \in E^*_{\vec{e}}} g\left(t, \overrightarrow{e_{opt}}, \vec{w}\right)\right)$$

Im konkreten Fall konnte nun das Problem, welches bei variablen Tarifeingaben auftritt, auf die Suche nach einem globalen Extremwert reduziert werden. In solchen Fällen spricht man von einer skalaren Optimierung. Hierbei ist aber noch anzumerken, dass es sich bei speziellen Extremwertermittlungen um ein Teilweise ungelöstes Problem handelt.

Bei einfachen differenzierbaren Funktionen ist das Bestimmen dieser Extremwerte mit geringem Aufwand möglich. So können zum Beispiel Tariflogiken, die auf einfache quadratische Gleichungen aufgebaut sind, einfacher behandelt werden, als komplexere nichtlineare Tariflogiken, bei denen man auf umfangreichere Methoden der globalen nichtlinearen Optimierung zurückgreifen muss. Vertreter hiervon sind zum Beispiel „Simulierte Abkühlung“ (simulated annealing), Metropolisalgorithmus, Schwellenakzeptanz (threshold accepting), usw. **[6].**

2.2.3.2 *Optimierung anhand des Eingabevektors*

Im vorangegangenen Kapitel konnten unter Zuhilfenahme einer Apriori Methode (hierbei handelt es sich um s.g. Skalarisierung) die variablen Elemente des Eingabevektors bestimmt werden. In diesem Kapitel wird nun ein weiterer Weg

beschrieben, wie man trotz variabler Elemente eine optimale Tarifwahl treffen kann. Unter der Annahme, dass wir einen Ausgabevektor haben, bei dem die einzelnen Vektorelemente aus linearen bzw. nicht linearen Termen bestehen und auch die Möglichkeit existiert, dass sich die besagten Vektorelemente beschränken lassen, dann besteht auch die Möglichkeit, ein lineares bzw. nichtlineares Optimierungsverfahren anzuwenden. In diesem Fall würde man von einer linearen Programmierung (LP) bzw. einer nicht linearen Programmierung (NLP) sprechen.

Unterstützend für eine LP bzw. NLP ist, wie bereits erwähnt, die Möglichkeit der Entwicklung von Nebenbedingungen (Eingangs erwähnt als Beschränkungen). Im Konkreten bedeutet das, dass man die variablen Elemente des Ausgabevektors als mathematische Terme für Nebenbedingungen verwendet. Diese Terme müssen als Gleichung bzw. Ungleichung definiert bzw. beschränkt werden. Im Falle einer linearen Programmierung müssen diese Gleichungen dem Namen entsprechend linear sein. Bei einer nichtlinearen Programmierung folglich nichtlinear **[7].**

Mit dem unten angeführten Ausgabevektor könnte man exemplarisch folgende Nebenbedingungen erzeugen.

$$\vec{e}_1 = \begin{pmatrix} 300 \\ x_1 \\ x_2 \end{pmatrix}$$

$$t^{(1)}_{\vec{k}_1}(\vec{e}_1) = \begin{pmatrix} 25 \\ 4x_1 + 3x_2 \\ x_1 - x_2 \end{pmatrix}$$

$Nb_1\colon\ 4x_1 + 3x_2 < 100 \quad Nb_2\colon\ x_1 - x_2 < 10 \quad Nb_3\colon\ x_1 \geq 0 \quad Nb_4\colon\ x_2 \geq 0$

Die oben angeführten Nebenbedingungen Nb_1 bis Nb_4 beschränken nun die möglichen Ausgabevektoren. Nur durch diese Nebenbedingungen ist es aber nur selten möglich, die variablen Elemente des Eingabevektors zu bestimmen. Ein derartiger Fall wäre zum Beispiel, dass sich alle Nebenbedingungen nur in einem Punkt schneiden und die passenden Gleichheits- bzw. Ungleichheitsbe-

dingungen besitzen. Im Wesentlichen heißt das, dass man noch ein Optimierungskriterium einführen muss. Im Falle des definierten Tarifmodells könnte man noch den Eingabevektor hinzuziehen und seine Vektorelemente parametrisieren. Diese Parametrisierung könnte durch ein Skalarprodukt des Eingabevektors mit einem Gewichtungsvektor abgebildet werden. Der resultierende Term wäre nun die Zielfunktion, die zu maximieren bzw. zu minimieren ist. Da die Optimierung nur die variablen Teile des Eingabevektors betrifft, müssen nicht alle Vektorelemente herangezogen werden. Anhand des oben angeführten Beispiels könnte das Skalarprodukt des Eingabevektors $\vec{e}_1$ mit einem Gewichtungsvektor $c \in \mathbb{R}^n$ wie folgt aussehen.

$$\begin{pmatrix} 300 \\ x_1 \\ x_2 \end{pmatrix} \quad \rightsquigarrow \quad \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}^T * \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = c_1 x_1 + c_2 x_2$$

Dieses Skalarprodukt kann nun verwendet werden, um die optimalen Werte der Variablen x_1 und x_2 zu finden. Formal könnte man die Optimierung auch als Maximierung darstellen.

$$\max\left(c_1 x_1 + c_2 x_2 \,\middle|\, \bigwedge_{n=1}^{4} Nb_n \right)$$

Die verwendeten Parameter c_1 und c_2 können einerseits als Gewichtung genutzt werden und andererseits können sie auch zur Bestimmung herangezogen werden, ob sich die parametrisierte Variable minimieren oder maximieren soll. Würde man das oben beschriebene Beispiel in einem Graphen darstellen (siehe Abbildung 2), so gäbe es fünf Extrempunkte (blau markiert). Je nach Wahl der Parameter c_1 und c_2 verändert sich die Position des optimalen Punktes.

Betrachtet man die Zielfunktion $c_1 x_1 + c_2 x_2$, so entspricht diese einem mathematischen Term einer Geraden in einem Koordinatensystem mit den Achsen x_1 und x_2. Bei einem Gewichtungsvektor $\vec{c} = \begin{pmatrix} 0{,}5 \\ -0{,}5 \end{pmatrix}$ bzw. $\vec{c} = \begin{pmatrix} -0{,}5 \\ 0{,}5 \end{pmatrix}$ wäre die Gerade wie Folgt definiert.

$$g_1 = \frac{x_1}{2} - \frac{x_2}{2}$$

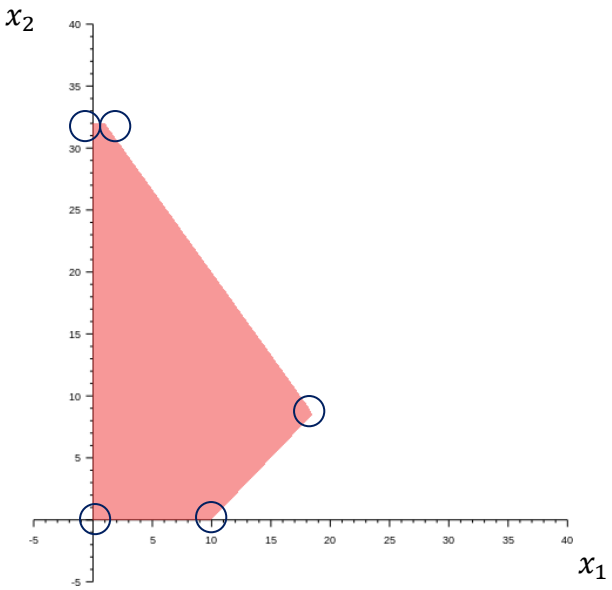


Abbildung 2 Darstellung der Nebenbedingungen

Die Suche nach dem Optimum würde nun einer parallelen Verschiebung dieser Geraden entsprechen (siehe Abbildung 3).

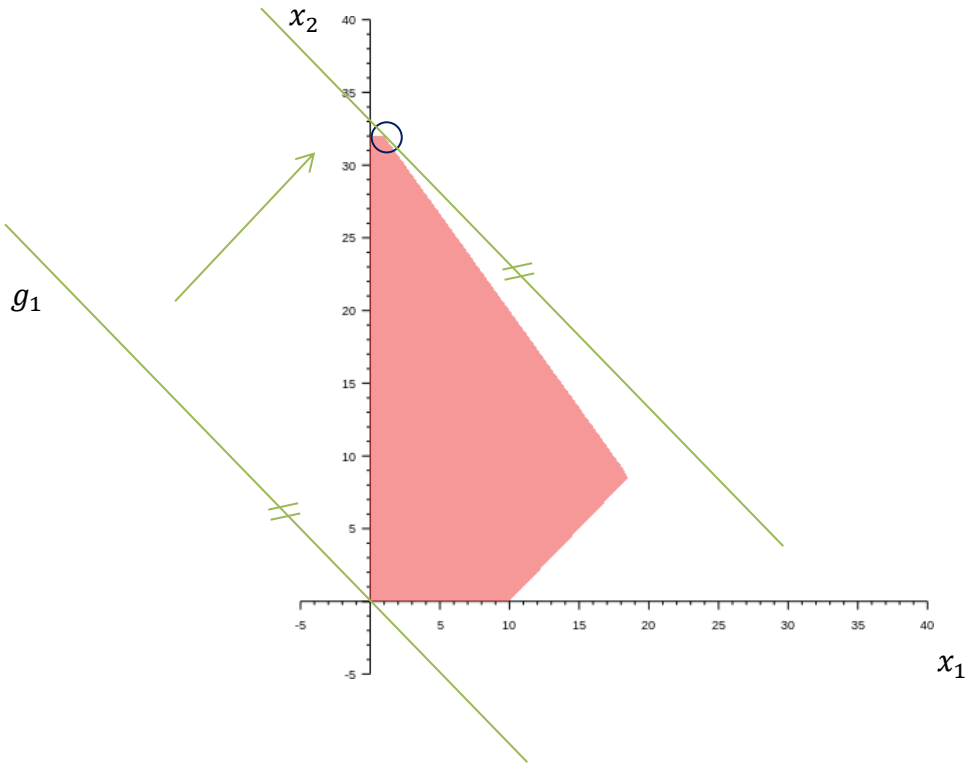


Abbildung 3 Suche nach dem optimalen Extrempunkt

Die angesprochene Verschiebung der Geraden entspricht aus mathematischer Sicht einer positiven bzw. negativen additiven Laufvariablen l in der Geradengleichung.

$$g_1 = \frac{x_1}{2} - \frac{x_2}{2} + l\ .$$

Das Optimum ist folglich der Extrempunkt, welcher durch das Inkrementieren bzw. Dekrementieren (abhängig, ob nach einem Minimum bzw. Maximum als Optimum gesucht wird) der Laufvariablen l als letztes erreicht wird (siehe

Abbildung 3 Suche nach dem optimalen Extrempunkt

- blaue Kreismarkierung). Bei einem passenden Gewichtungsvektor $\vec{c} \in \mathbb{R}^l$: $l \in \mathbb{N}$ ist es aber auch möglich, dass es keine eindeutige Lösung gibt. In diesem Fall würde mehr als ein Punkt auf der Geraden der Zielfunktion liegen (siehe Abbildung 4 - blaues Rechteck). In solchen Fällen könnte man in der Praxis eine interaktive Anpassung des Gewichtungsvektors vornehmen.

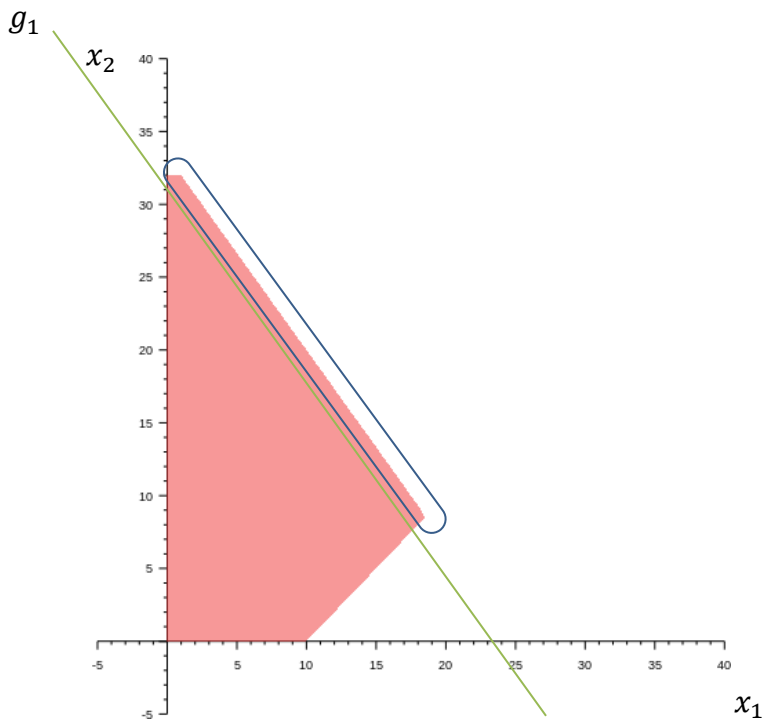


Abbildung 4 Gewichtungsvektor mit keiner eindeutigen Lösung

Das beschriebene Verfahren liefert nun die nach Nebenbedingungen beschränkten optimalen variablen Elemente des Einagevektors. Anzumerken ist aber, dass das angeführte Beispiel, bedingt durch nur zwei unbekannte Variablen, sehr gut grafisch gelöst werden konnte. Das muss nicht immer der Fall sein. In der Praxis kann es vorkommen, dass man sich mit einer größeren Anzahl an unbekannten Variablen beschäftigen muss. Daher ist es sehr wichtig, dass man auf passende LP- bzw. NLP- Lösungsverfahren zurückgreift. Das Simplexverfahren(LP), die Ellipsoidmethode(LP) bzw. die Quadratisches Programmierung(NLP) sind nur einige der möglichen Verfahren, die man anwenden kann.

Da das oben beschriebene Beispiel sehr exemplarisch aufgebaut ist, und auch die Möglichkeit existiert, dass das Optimierungskriterium komplexerer Natur ist, sollte man abschließend noch eine generelle Definition formalisieren.

Allgemein kann man daher sagen, dass man bei einer beliebigen linearen bzw. nichtlinearen skalaren Zielfunktion

$$z : (\mathbb{R}^m, \mathbb{R}^l) \rightarrow \mathbb{R} \qquad \forall\, m, l \in \mathbb{N}$$

mit einem Eingabevektor $\vec{e} \in \mathbb{R}^m$, sowie einem Konfigurationsvektor $\overrightarrow{c} \in \mathbb{R}^l$ (beim Skalarprodukt als Zielfunktion wurde er auch als Gewichtungsvektor bezeichnet) und den Nebenbedingungen Nb_1 ... Nb_n , $n \in \mathbb{N}$ die Optimierung formal wie folgt definieren kann.

$$\min\left(z(\vec{e}, \vec{c}) \,\middle|\, \bigwedge_{x=1}^{n} Nb_x \right)$$

Analog zum Kapitel 2.2.3.1 ist es nun auch noch wichtig, dass definiert wird, wie das optimale Tariftupel gefunden werden kann. Daher sei $n \in \mathbb{N}$ die Anzahl der Tariflogiken und $\overrightarrow{k_{(y)}^{(x)}}$ eine zu $t^{(x)}$ gehörige Tarifkonfiguration aus der Menge der möglichen Tarifkonfigurationen $K^{(x)}$, wobei gilt, dass $x, y \in \mathbb{N}$ ist. Die Menge aller möglichen Tariftupel T kann nun wie folgt definiert werden.

$$T = \left\{ \left(t^{(x)}, \overrightarrow{k_{(y)}^{(x)}} \right) \,\middle|\, (x \geq 1 \wedge x \leq n) \wedge \left(y \geq 1 \wedge y \leq \left| K^{(x)} \right| \right) \wedge \overrightarrow{k^{(x)}} \in K^{(x)} \right\}$$

Sei I die Indexmenge eines konkreten Eingabevektors $\vec{e}$, welche beschränkt auf die nicht variablen Vektorelemente ist, dann ist Menge der möglichen Eingabevektoren $E_{\vec{e}}^{*}$ welche dem konkreten Eingabevektors $\vec{e}$ zugrunde liegen, wie folgt definiert.

$$E_{\vec{e}}^{*} = \{\vec{x} \mid \vec{x} \in E \ \wedge \ \forall\, i \in I : x_i = e_i \ \} \ \Rightarrow \ E_{\vec{e}}^{*} \subseteq E : \vec{e} \ \in E$$

Aufbauend kann man nun sagen, dass bei einem Vektor $\overrightarrow{e_{opt}} \ \in \ E_{\vec{e}}^{*}$ die Tariflogikfunktion $t_{\underset{k^*}{\rightarrow}}^{*}$ genau dann optimal ist, wenn folgendes gilt.

$$t_{\underset{k^*}{\rightarrow}}^{*}(\vec{e}) = \min_{t \in T}\left(\min_{\overrightarrow{e_{opt}} \in E_{\vec{e}}^{*}}\left(z(\vec{e},\vec{c}) \,\middle|\, \bigwedge_{x=1}^{n} Nb_x\right)\right)$$

2.2.3.3 *Variabilität der Eingabevektorbestimmung*

In den beiden vorangegangen Kapiteln wurden zwei Möglichkeiten beschrieben, wie man trotz einer nicht vollständigen Tarifeingabe ein optimales Tariftupel ermitteln kann. Dennoch wurde ein wichtiger Punkt nicht erörtert. Im Konkreten geht es hierbei um den Fall, dass es durchaus möglich sein kann, dass bei der Ermittlung der variablen Eingabevektorelemente durch die beiden beschriebenen Verfahren kein einheitlicher optimaler Eingabevektor ermittelt wird.

2.2.3.3.1 Formalisierung und Beweis

Formal könnte man die oben beschriebene Aussage nun wie folgt ausdrücken.

Sei $\vec{e}$ ein Eingabevektor aus der Menge der möglichen Eingabevektoren E . Des Weiteren bestehe dieser Eingabevektor $\vec{e}$ aus variablen Vektorelementen, sodass es eine zugehörige Menge $E_{\vec{e}}^{*} \ \subseteq \ E$ gibt, die alle Eingabevektoren beinhaltet, welche aus dem Eingabevektor $\vec{e}$ gebildet werden können. Ebenso sei das anwendbare Optimierungsverfahren zur Ermittlung des optimalen Eingabevektors vereinfacht als folgende Funktion dargestellt.

$$OPT\colon (T, E) \ \rightarrow \ E^{*}$$

Die eingangs erwähnte Aussage entspricht nun folgender formalen Behauptung

$$Es\ existiert\ eine\ Tariftupelmenge\ T, f\ddot{u}r\ die\ gilt, dass$$

$$P(\ \forall x \in O_{\vec{e}} : \ \forall y \in O_{\vec{e}} \setminus \{x\} : x = y\ ) \leq 1$$

$$unter\ der\ Voraussetzung, dass$$

$$O_{\vec{e}} = \{OPT(t_x, \vec{e}) \mid x \geq 0 \wedge x \leq |T| \wedge t_x \in T \wedge \vec{e} \in E \wedge x \in \mathbb{N}\ \}$$

Anm.: $O_{\vec{e}}$ ist die Menge aller optimalen Eingabevektoren, die man aus der Menge T und dem Eingabevektor $\vec{e}$ bilden kann.

Um diese Aussage zu belegen, muss ein Existenzbeweis durchgeführt werden. Hierzu ist es hilfreich, wenn man die Hauptaussage in zwei Fälle separiert. Wenn gezeigt werden kann, dass beide dieser Fälle existieren, so kann man auch die getätigte Aussage bestätigen.

Fall 1: $P(\ \forall x \in O_{\vec{e}} : \ \forall y \in O_{\vec{e}} \setminus \{x\} : x = y\ ) = 1$

Für diesen Fall ist es ausreichend zu zeigen, dass bei einer exemplarischen Menge an Tariftupeln T, und einem konkreten Eingabevektor $\vec{e}$ mit variablen Elementen nur ein optimaler Eingabevektor $\overrightarrow{e_{opt}}$ existiert.

Man definiere daher zunächst den konkreten Eingabevektor$\vec{e}$.

$$\vec{e} = \begin{pmatrix} z \\ 3 \end{pmatrix}, \qquad z \in \mathbb{R}^+$$

Des Weiteren wird die Menge T wie folgt definiert.

$$T = \left\{\left(t^{(1)}_{\overrightarrow{k_1}}, \overrightarrow{k_1}\right), \left(t^{(2)}_{\overrightarrow{k_2}}, \overrightarrow{k_2}\right)\right\}$$

$$t^{(1)}_{\overrightarrow{k_1}}(\vec{e}) = \begin{pmatrix} \frac{1}{z^3} + 3 \\ z^2 \end{pmatrix} \qquad t^{(2)}_{\overrightarrow{k_2}}(\vec{e}) = \begin{pmatrix} z^2 \\ \frac{1}{z^3} + 3 \end{pmatrix}$$

Für das Ermitteln der optimalen Eingabevektoren, bietet sich die Skalarisierung sehr gut an. Das Optimierungsverfahren OPT wäre somit eine Kurvendiskussion mit den skalarisierten Ausgabevektoren. Die Optimierung muss daher in zwei Schritten aufgeteilt werden.

1. Skalarisieren des Ausgabevektors

Zur Skalarisierung benötigen wir zunächst einen Gewichtungsvektor $\vec{w} \in \mathbb{R}^2$.

$$\vec{w} = \begin{pmatrix} 0{,}5 \\ 0{,}5 \end{pmatrix}$$

Dieser ist ein wichtiger Teil der Mappingfunktion $g(t,\vec{e},\vec{w})$ (siehe Kapitel 2.2.3.1).

$$g(t_1,\vec{e},\vec{w}) = g(t_2,\vec{e},\vec{w}) = 0{,}5 * \left(\frac{1}{z^3} + 3\right) + 0{,}5z^2, \qquad t_1, t_2 \in T$$

Hier kann man schon sehr gut erkennen, dass man auch bei zwei unterschiedlichen Tariftupeln idente optimale Eingabevektoren erhalten kann. Dennoch sollte zur Vollständigkeit auch noch die Optimierung illustriert werden.

2. Diskussion der Mappingfunktion

Die ermittelte Mappingfunktion wird in diesem Schritt auf ihre Minimalwerte analysiert. Im konkreten Fall kann dies durch eine einfache Kurvendiskussion gemacht werden. Dafür müssen alle lokalen Maxima/Minima anhand der ersten Ableitung ermittelt werden.

$$\frac{\partial}{\partial z}\ g(t_1,\vec{e},\vec{w}) = \frac{\partial}{\partial x}\ g(t_2,\vec{e},\vec{w}) = z - \frac{3}{2z^4} = 0 \qquad \longrightarrow \qquad z_0 = \sqrt[5]{\frac{2}{3}}$$

Da es sich bei z_0 um das einzige Minimum im Bereich von $\mathbb{R}^+$ handelt ($g'(t_1,\vec{e},\vec{w}) > 0$ und $g'(t_2,\vec{e},\vec{w}) > 0$), besitzt diese Optimierung eine eindeutiges Ergebnis.

$$\overrightarrow{e_{opt}^{t_1}} = \overrightarrow{e_{opt}^{t_2}} = \begin{pmatrix} \sqrt[5]{\frac{2}{3}} \\ 3 \end{pmatrix}$$

Die Menge $O_{\vec{e}}$ wäre daher wie folgt aufgebaut.

$$O_{\vec{e}} = \left\{\overrightarrow{e_{opt}^{t_1}}, \overrightarrow{e_{opt}^{t_2}}\right\} = \left\{\begin{pmatrix} \sqrt[5]{\frac{2}{3}} \\ 3 \end{pmatrix}, \begin{pmatrix} \sqrt[5]{\frac{2}{3}} \\ 3 \end{pmatrix}\right\}$$

$$P(\,\forall x \in O_{\vec{e}} : \forall y \in O_{\vec{e}} \setminus \{x\} : x = y\,) = 1$$

Die zu beweisende Aussage ist somit erfüllt.

Fall 2: $P(\,\forall x \in O_{\vec{e}} : \forall y \in O_{\vec{e}} \setminus \{x\} : x = y\,) < 1$

Für diesen Fall muss man zeigen, dass es bei einer exemplarischen Menge an Tariftupeln T und einem konkreten Eingabevektor $\vec{e}$ mit variablen Elementen unterschiedliche optimale Eingabevektoren gibt. Hierfür kann man das Beispiel vom *Fall 1* aufgreifen und anpassen.

Sei nun die Menge der Tariftupel T und der Eingabevektor $\vec{e}$ ident definiert wie bei *Fall 1.* Des Weiteren sei auch das Optimierungsverfahren OPT als Kurvendiskussion der skalarisierten Ausgabevektoren gewählt. Diese Annahmen implizieren folglich, dass man auch hier in zwei Schritten die optimalen Eingabevektoren bestimmen kann.

1. *Skalarisieren des Ausgabevektors*

   Zur Skalarisierung benötigen wir ebenfalls zunächst den Gewichtungsvektor $\vec{w} \in \mathbb{R}^2$.

$$\vec{w} = \begin{pmatrix} 0{,}25 \\ 0{,}75 \end{pmatrix}$$

   Daraus lassen sich nun folgende zwei Mappingfunktionen erstellen.

$$g(t_1,\vec{e},\vec{w}) = 0{,}75 * \left(\frac{1}{z^3} + 3\right) + 0{,}25z^2, \qquad t_1 \in T$$

$$g(t_2,\vec{e},\vec{w}) = 0{,}25 * \left(\frac{1}{z^3} + 3\right) + 0{,}75z^2, \qquad t_2 \in T$$

Hierbei kann man schon sehr gut erkennen, dass man nur durch die Änderung des Gewichtungsvektors aus dem Beispiel des ersten Falles, eine veränderte Ausgangslage der Mappingfunktions Diskussion bewirken kann.

2. Diskussion der Mappingfunktion

Die Analyse der beiden Mappingfunkionen entspricht der des ersten Falles.

$$\frac{\partial}{\partial z}\ g(t_1,\vec{e},\vec{w}) = \frac{z}{2} - \frac{9}{4z^4} = 0 \qquad \longrightarrow \qquad z_0^{\#} = \frac{3^{2/5}}{\sqrt[5]{2}}$$

$$\frac{\partial}{\partial x}\ g(t_2,\vec{e},\vec{w}) = \frac{3z}{2} - \frac{3}{4z^4} = 0 \qquad \longrightarrow \qquad z_0^{*} = \frac{1}{\sqrt[5]{2}}$$

Da es sich bei $z_0^{\#}$ und z_0^{*} um die beiden einzigen Minima im Bereich von $\mathbb{R}^+$ handelt ($g'(t_1,\vec{e},\vec{w}) > 0$ und $g'(t_2,\vec{e},\vec{w}) > 0$), besitzt diese Optimierung zwei eindeutige Ergebnisse. Die Menge $O_{\vec{e}}$ wäre daher wie Folgt aufgebaut.

$$O_{\vec{e}} = \left\{\overrightarrow{e_{opt}^{t_1}}, \overrightarrow{e_{opt}^{t_2}}\right\} = \left\{\begin{pmatrix} \frac{3^{2/5}}{\sqrt[5]{2}} \\ 3 \end{pmatrix}, \begin{pmatrix} \frac{1}{\sqrt[5]{2}} \\ 3 \end{pmatrix}\right\}$$

$$P(\ \forall x \in O_{\vec{e}} : \ \forall y \in O_{\vec{e}} \setminus \{x\} : x = y\ ) = 0 < 1$$

Die zu beweisende Aussage ist somit erfüllt.

2.2.3.3.2 Folgerungen

Da nun bewiesen wurde, dass es bei der Ermittlung der optimalen Eingabevektoren durchaus vorkommen kann, dass die Optimierungsverfahren keine einheitlichen Vektoren ermitteln, muss man die oben besprochenen Verfahren erweitern, um überhaupt ein optimales Tariftupel finden zu können. Im Allgemeinen münden diese Erweiterungen in einem Vektoroptimierungsproblem, bei dem kombinierte Vektoren aus Eingabe- und Ausgabewerten hinsichtlich ihrer Optimalität untersucht werden. Die nun beschriebenen Methoden unterscheiden sich hinsichtlich der der Ausgangslage beim Lösen des Vektoroptimierungs-

problems. Bei der ersten Methode wird die Dimension des Vektoroptimierungsproblems nur um eine Ebene erhöht, wohingegen bei der zweiten Methode die Vektoroptimierungsdimension um die Anzahl der Zeilen im Ausgangsvektor erhöht wird.

Methode 1:

Betrachtet man das erste Verfahren (Optimierung anhand des Ausgabevektors – Kapitel 2.2.3.1), so bestünde unter anderem die Möglichkeit, das entwickelte Skalarprodukt als Vektorelement im optimalen Eingabevektor zu integrieren. Somit würde das Vektoroptimierungsproblem mit der zugrunde liegenden Suche nach dem besten Tariftupel um nur eine zusätzliche Dimension erweitert werden.

Wenn T die Menge aller Tariftupel, die Funktion g die Mapping- bzw. Skalarisierungsfunktion (siehe Kapitel 2.2.3.1) für eine beliebiges Tariftupel $t_x \in T$ und OPT die Optimierungsfunktion zur Bestimmung des optimalen Eingabevektors für einen Eingabevektor $\vec{e}$ und ein konkretes Tariftupel aus der Menge T ist, dann kann die Ergebnismenge R (siehe Kapitel 2.2.2) wie folgt definiert werden.

$$R_{\vec{e}} = \left\{ \begin{pmatrix} o_1 \\ \vdots \\ o_n \\ g(t_x, \vec{o}, \vec{w}) \end{pmatrix}^{(t_x)} \middle| \begin{array}{c} d, n, x \in \mathbb{N} \\ x \geq 0 \wedge x \leq |T| \\ t_x \in T \\ \vec{e} \in E \\ \vec{w} \in \mathbb{R}^d \\ \vec{o} \in \mathbb{R}^n \\ \vec{o} = OPT(t_x, \vec{e}) \end{array} \right\}$$

Zur Bestimmung des optimalen Tariftupels können nun unter anderem die im Kapitel 2.2.2 angesprochenen Verfahren verwendet werden.

Diese Methode ist auch beim zweiten angeführten Verfahren (Optimierung anhand des Eingabevektors – Kapitel 2.2.3.2) anwendbar und unterscheidet sich nur durch das Auswechseln der Optimierungsfunktion OPT.

Methode 2:

Bei der zweiten Methode wird nicht wie bei der ersten Methode ein Skalar als Abbildung des Ausgabevektors zum optimalen Eingabevektor hinzugefügt. Diese Methode verwendet den gesamten Ausgabevektor für die Bestimmung des besten Tariftupels.

Wenn T die Menge aller Tariftupel und OPT die Optimierungsfunktion zur Bestimmung des optimalen Eingabevektors für einen Eingabevektor $\vec{e}$ und ein konkretes Tariftupel aus der Menge T ist, dann kann die Ergebnismenge R wie folgt definiert werden.

$$\vec{h}\colon (T,E) \rightarrow \mathbb{R}^d \qquad h(t,\vec{e}) = f_{t_2}(\vec{e}) \,:\, f \equiv t_1 \wedge d \in \mathbb{N}$$

$$R_{\vec{e}} = \left\{ \begin{pmatrix} o_1 \\ \vdots \\ o_n \\ a_1 \\ \vdots \\ a_m \end{pmatrix}^{(t_x)} \;\middle|\; \begin{array}{c} n, m, x \in \mathbb{N} \\ x \geq 0 \wedge x \leq |T| \\ t_x \in T \\ \vec{e} \in E \\ \vec{o} \in \mathbb{R}^n \\ \vec{a} \in \mathbb{R}^m \\ \vec{o} = OPT(t_x, \vec{e}) \\ \vec{a} = h(t_x, \vec{e}) \end{array} \right\}$$

Auch hier können zur Bestimmung des optimalen Tariftupels alle Verfahren verwendet werden, die bereits im Kapitel 2.2.2 angesprochen wurden.

2.2.4 Zusammenfassung

In den vorangegangen Kapiteln wurde auf die Problematiken einer Tarifempfehlung eingegangen. Anfänglich wurde formal definiert, was eine Tarifempfehlung ist. Anhand dieser Definition wurden anschließend zwei grundsätzliche Fälle einer Tarifempfehlung ermittelt. Diese Anwendungsfälle wurden infolge dessen mit Hilfe des mathematischen Formalismus beschrieben. Diese Beschreibungen sollten ein Bild der Komplexität liefern und auch zeigen, womit man bei einer praktischen Umsetzung konfrontiert werden könnte.

Grundsätzlich wurden die besprochenen Anwendungsfälle auf Basis des Eingabevektors kategorisiert (Vollständigkeit des Eingabevektors). Beide Fälle mündeten aber schlussendlich in einem Vektoroptimierungsproblem. Hinsichtlich der Komplexität konnte man sehr gut erkennen, dass diese mit Anstieg der Vektordimension auch zunahm. Ebenfalls konnte festgestellt werden, dass die Möglichkeit von Tarifkompositionen ein hohes Maß an Komplexität mit sich bringt und eine hohe praktische Relevanz birgt. Gewichtig hinsichtlich der praktischen Umsetzung sind aber auch die nötigen Optimierungsverfahren, die speziell im zweiten Anwendungsfall (UC2) von Nöten sind, da viele kongruente Optimierungsprobleme zum Teil ungelöste Probleme sind (z.B. Suche nach dem globalen Minimum). Bei einer praktischen Umsetzung sollte man daher speziell beim Thema der Tarifempfehlung auf optimierte Algorithmen und spezialisierte Softwarebibliotheken setzen. Wie man nun Tarifempfehlungen in eine Softwarelösung einbauen kann, wird im nächsten Kapitel analysiert.

2.3 Tarife in Hochsprachen

Anhand des definierten Tarifmodells vom Kapitel 2.1, sowie den darin durchgeführten Analysen, wird in diesem Kapitel versucht, das formal beschriebene Modell hinsichtlich der Abbildbarkeit durch eine objektorientierte Hochsprache zu erläutern. Im Wesentlichen sollte mit Hilfe der objektorientierten Programmiersprache Java und der Modellierungssprache UML(Unified Modeling Language) ein Konzept erarbeitet werden, das demonstrieren soll, wie das geschilderte Modell auf einer maximalen Abstraktionsebene implementiert werden kann.

Das entworfene Konzept kann anschließend auch als Vorlage für weitere Hochsprachen angesehen werden. Zudem sollte dieses Konzept auch als Grundlage für das konzeptionelle Programmframework für KMUs – beschrieben im nächsten Kapitel - dienen.

Im Zuge der Projektarbeit wurde zudem auch eine Programmbibliothek für die Programmiersprache Java entwickelt, welche das entworfene Konzept praktisch umsetzten sollte. Unter dem Namen JavaRates findet man im Package `at.tugraz.ist.javarates` die entworfene Referenzimplementierung.

In den nächsten Kapiteln werden daher anfänglich offensichtliche Schlüsselprobleme definiert und in weiterer Folge ermittelt, wie sie mit einer objektorientierten Programmiersprache bestmöglich gelöst werden können.

2.3.1 Schlüsselprobleme

Formale mathematische Modelle müssen in der Regel konkretisiert werden, damit sie für Hochsprachen beschreibbar sind. Genau diese Konkretisierungen, sowie die Voraussetzung, dass in einer hoch abstrahierten Modellierung aktuelle und zukünftige Eventualitäten bestmöglich implementiert und abgebildet sein sollten, bilden im Wesentlichen die Hauptproblematik. Diese Ausgangslage, sowie die Vielfalt an Tarifen, machen es schwer eine solche Modellierung zu entwickeln. Zusätzlich gibt es auch noch die Einschränkung durch Programmierparadigmen, die auch berücksichtigt werden müssen, aber nicht fundamentaler Natur sind.

Um nun eine solche Modellierung konstruieren zu können, sollten zunächst mal die Hauptkomponenten des im Kapitel 2.1 beschriebenen Modells betrachtet werden. Diese drei Komponenten müssen primär hinsichtlich ihrer Abbildbarkeit durch eine Hochsprache analysiert werden. Im folgenden Abschnitt wird daher auf Kernpunkte (K) eingegangen, die anhand der angestellten Analysen vorrangig sind. Weitere Punkte mit geringerem Stellenwert, s.g. Probleme (P), werden im nächsten Unterkapitel im Zuge der Beschreibung der gewählten Architektur angeführt und erläutert.

Da in diesem Abschnitt eine objektorientierte Hochsprache und die Modellierungssprache UML als Modellierungshilfen angewendet werden, sollte man die auftretenden Modellkomponenten in Klassen zusammenfassen. *Tarifeingaben, Tarifausgaben* und *Tarifkonfigurationen* können auf Grund ihrer Ähnlichkeit in verwandten Klassenobjekten gekapselt werden. Da die T*ariflogik* den Kern des Tarifmodells bildet, sollte sie in einer einzelnen Klasse abgebildet werden.

Betrachtet man nun diese Klassen, so müssen folgende drei grundlegende Punkte berücksichtigt werden, um eine maximale Abstraktion zu erzielen.

K1. Der im Tarifmodell beschriebene Eingabevektor / Ausgabevektor / Konfigurationsvektor ist ein n-Tupel reeller Zahlen. Demzufolge müssen die in den Eingabeobjekten / Ausgabeobjekten / Konfigurationsobjekten gekapselten Eingabewerte / Ausgabewerte / Konfigurationswerte auf Gleitkommazahlen aufbauen und eine ausreichende Genauigkeit liefern.

K2. Da es sich bei der Tarifeingabe / Tarifausgabe / Tarifkonfiguration um einen variablen Vektor handelt (Anzahl der Vektorelemente kann sich je nach Tarif ändern), muss ein Eingabeobjekt / Ausgabeobjekt / Konfigurationsobjekt auch eine variable Anzahl an Parametern besitzen.

Zu den bereits genannten Punkten gibt es im Falle der Tariflogikklasse noch einen weiteren Punkt, der zwingen erfüllt und bedacht werden muss.

K3. Tariflogiken sind einem Anpassungsprozess unterworfen und müssen daher im Zuge ihrer Verwendung in einer Implementierung (zukünftig auch als Verwaltungsprogramm bezeichnet) auch austauschbar sein. Eine Modellierung bzw. die verwendete Architektur sollte daher einen flexiblen Logikaustausch besitzen bzw. ermöglichen.

Alle oben genannten Punkte sollten bei einer Umsetzung des formal definierten Tarifmodels anhand einer objektorientierten Hochsprache beachtet werden. Demgemäß muss auch die Architektur diesen Voraussetzungen angepasst sein.

2.3.2 Objektorientierte Umsetzung

In diesem Kapitel wird nun beschrieben, wie die oben genannten Punkte hochabstrahiert implementiert werden können. Dafür wird mit Hilfe der Modellierungssprache UML eine Architektur illustriert, die sich der Kernproblematiken annimmt. Unterstützend werden anhand der Programmiersprache Java interessante Programmcodeteile angeführt. Diese sollten für ein besseres Verständnis hinsichtlich der auftretenden Kernprobleme sorgen.

2.3.2.1 *Abbildung der Eingabe- / Ausgabe- / Konfigurationsvektoren*

Die im vorangegangen Kapitel definierten Kernprobleme K1 und K2 bilden den Ausgangspunkt für die nun beschriebene objektorientierte Umsetzung. Das UML-Klassendiagramm (siehe Abbildung 5) beinhaltet den gewählten Lösungsansatz der Kernprobleme K1 und K2.

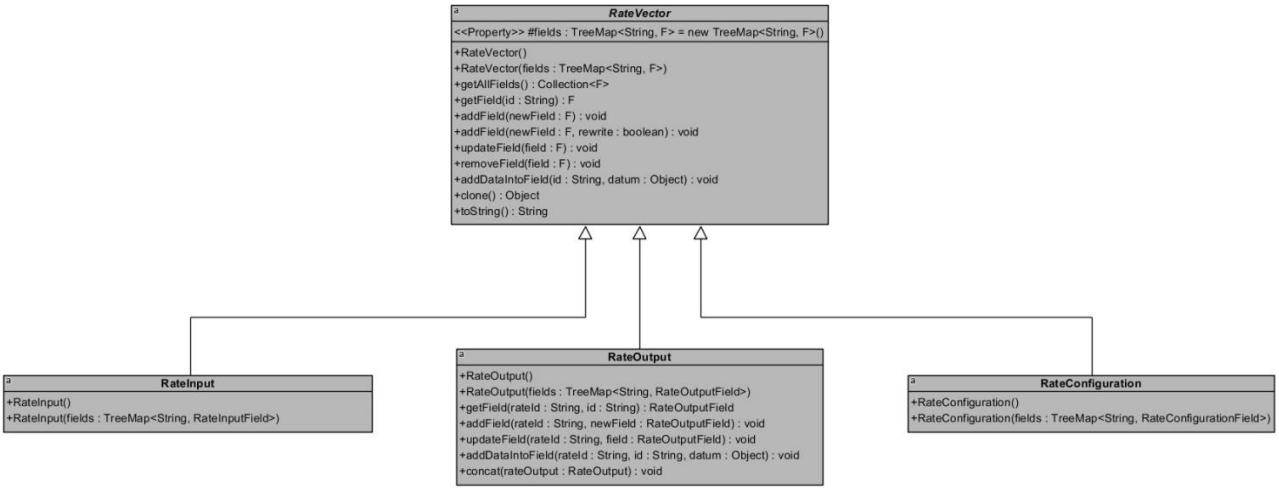


Abbildung 5. UML-Klassendiagram / Tarifeingaben, -ausgaben und –konfigurationen

Im Fall von K1 wird eine Datenstruktur eingeführt, welche die Variabilität eines n-Tupels bestmöglich abbildet. Für ein besseres Ansprechen der n-Tupelwerte sollte ein assoziatives Datenfeld gewählt werden. Da wir in diesem Abschnitt für demonstrative Zwecke Java verwenden, kommt hierfür eine Implementierung des `java.util.Map<K,V>` Interfaces zur Anwendung.

Für den generischen Datentyp `V` sollte modellgemäß ein Fließkommadatentyp wie zum Beispiel `double` verwendet werden. Mit diesen Fließkommazahlen ist es bereits möglich, die nötigen Teilmengen der reellen Zahlen für die Eingabe- / Ausgabe- /Konfigurationsmenge abzubilden[19]**[8]**. Dennoch kann es in manchen Fällen sinnvoller sein, gewisse n-Tupelelemente als Objekte andere Datentypen zu übergeben. Daher wäre eine Kapselung dieser n-Tupelwerte durch eine weitere Klasse sinnvoller (`at.tugraz.ist.javarates.RateVectorField`). Zudem ist es dadurch auch möglich, Fleißkommadatentypen zu verwenden, die

[19] Zusammenfassung: Dipl. Math. Frank Braun, Uni Regensburg – „Binärformate reeller Zahlen“ http://homepages.uni-regensburg.de/~brf09510/float/realform.pdf - 18.11.2016

nicht als Standarddatentypen vorhanden sind. Klassen, wie zum Beispiel die Java-Klasse `java.math.BigDecimal`,bieten durch ihre Langzahlenarithmetik eine viel bessere Genauigkeit und eigenen sich daher weitaus besser für monetäre Berechnungen [20].

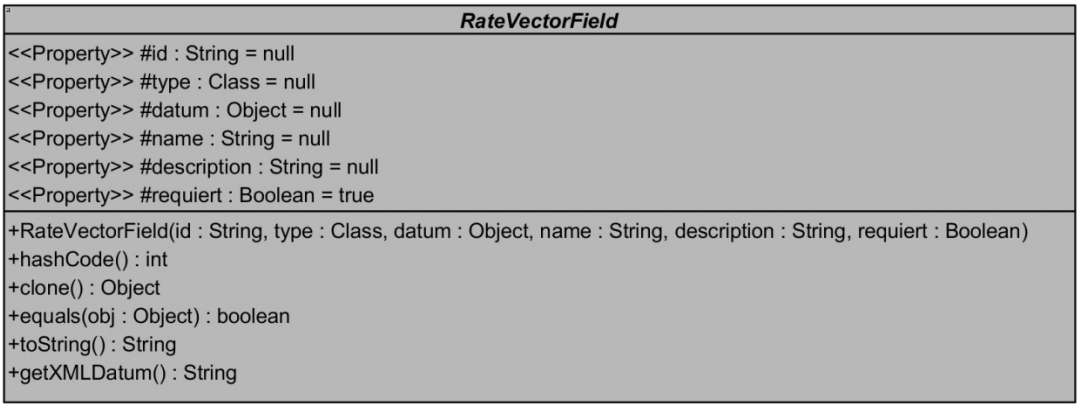


Abbildung 6. Vektorwerte als erweitertes Objekt

Aufgrund der oben beschriebenen Sachlage kann man im Klassendiagramm der Abbildung 6 sehr gut erkennen, dass alle Vektorwerte in maximal abstrahierten `java.lang.Object` Instanzen gespeichert werden. Um den konkreten Datentyp des n-Tupelwertes definieren zu können, findet man in der Kapselungsklasse auch ein Attribut des Datentyps `java.lang.Class`. Dieses Attribut ermöglicht es, dass das gespeicherte abstrakte Objekt im Zuge einer Verwendung durch die Tariflogik einer passenden Typumwandlung unterzogen werden kann.

2.3.2.2 *Abbildung der Tariflogik*

Da bisher nur die Eingabe-, Ausgabe- und Konfigurationsschnittstelle definiert wurde, wird anhand des nächsten Abschnitts beschrieben, wie eine Tariflogikinstanz aussehen muss, um damit der Modelvorgabe zu entsprechen. Die nun im Anschluss getroffenen Designentscheidungen behandeln das Kernproblem K3 und befinden sich primär auf der Architekturebene.

[20] JSR 354: Money and Currency API - https://jcp.org/en/jsr/detail?id=354 – 04.03.2017

Die im vorangegangenen Kapitel beschriebene Umsetzung der Tarifvektoren kann als grundsätzliche Einschränkung für die Umsetzung des Kernproblems K3 in diesem Kapitel gesehen werden. Das bedeutet im Wesentlichen, dass die in diesem Kapitel erarbeitete Architektur die im vorigen Kapitel definierten Objektklassen integrieren muss. Eine weitere Randbedingung mit großem Augenmerk ist die faktische Notwendigkeit, dass Tarife speicherbar sein müssen.

Da der Großteil der sich im Umlauf befindenden Tarife in Prosa definiert ist und nicht als eine mathematische Funktion greifbar ist, stellt sich nun im Kontext der oben beschriebenen Randbedingungen, sowie dem zu lösenden Kernproblem K3 folgende anfängliche Frage:

„Wie kann man prosaisch definierte Tarife und die darin enthaltenen Tariflogiken als Programmteile zugänglich machen?“

Wie bereits im Kapitel 2.1.3 sehr gut zu erkennen war, kann man prosaisch definierte Tarife durch geschicktes Transformieren zu einer mathematischen Funktion umwandeln. Eine automatisierte Transformation dieser mathematischen Funktion hin zu einer ausführbaren Logik kann sehr umständlich sein. Folglich wäre das Einbinden von Spezialsoftware ein möglicher Lösungsansatz. Eine derartige Softwarebibliothek (Application Programming Interface - API) müsste einen funktionellen Umfang haben, der es ermöglicht, die bereits zu mathematischen Formeln transformierten Tariflogiken im mathematischen Sinne zu lösen. Eine solche Spezialsoftware mit passenden Programmierschnittstellen wäre zum Beispiel WolframAlpha[21]. Dieser Ansatz kann aber als sehr schwergewichtig gesehen werden, da man in der Regel nur eine geringe Anzahl an mathematischen Operationen benötigt.

Der in diesem Projekt gewählte Ansatz basiert auf der Beobachtung, dass Programmiersprachen per se bereits eine Vielzahl an mathematischen Funktionen integriert haben[22]. Somit würde eine direkte Transformation von prosaischen Tarifen hin zu einem tarifabbildenden Programmcode ohne weitere Zwischen-

[21] www.wolframalpha.com

[22] Java Platform SE 8 Documentation – java.lang.Math

schritte möglich sein. Würden man das im Kapitel 2.1.3 angeführte Beispiel zu Pseudocode transformieren, könnte das Resultat wie in Abbildung 7 skizziert aussehen.

```
EINGABE:    stkBergeKFZ
            stkBergefachkraft
            stkBergehelfer
            stkAnschlagmittel
            stdBergeKFZ
            stdBergefachkraft
            stdBergehelfer
            reinigung

KONFIGURATION:    preise[0..4]
                  minVerrechnungsMenge[0..2]

AUSGABE:    kosten

function()
start

   kosten = 0

   if stkBergeKFZ > 0 then
      if stkBergeKFZ < stdBergeKFZ then
            kosten = kosten + (preise[0]*stkBergeKFZ)
      end if
      else
            kosten = kosten + (preise[0]*stdBergeKFZ)
      end else
   end if

   .
   .
   # Bergefachkraft und Bergehelfer wie bei BergeKFZ
   .
   .

   kosten = kosten + ( stkAnschlagmittel * preis[3] )
   kosten = ( reinigung == true )?( kosten + preis[4] ) : kosten

   return kosten

end
```

Abbildung 7. Pseudocode nach Transformation

Mit der Feststellung, dass man Programmcode verwenden kann, um prosaische Tarife abzubilden, ergibt sich auch eine Konkretisierung in Hinsicht auf die Softwarearchitektur. Im Grunde gibt es zwei Möglichkeiten schriftliche Tarife zu einem ausführbaren Programm zu konvertieren.

Möglichkeit 1: Hierbei wird der gewünschte Tarif mit Hilfe einer proprietären Programmiersprache definiert. Der transformierte Tarif wird anschließend in das Verwaltungsprogramm geladen und interpretiert. Da es sich dabei um eine eigens definierte Programmiersprache handelt, muss ein passender Compiler entwickelt werden, der das entwickelte Tarifprogramm hin zu einem ausführbaren Programmcode kompiliert.

Vorteil bei dieser Möglichkeit ist, dass man die proprietäre Syntax passend für Tarife praxisnahe entwickeln kann und grundsätzliche Tariflogikschemata abstrahiert abbilden kann. Nachteil hierbei ist der enorme zeitliche Aufwand für die Entwicklung dieser Syntax, sowie des nötigen Compilers.

Möglichkeit 2: Diese Möglichkeit bedient sich einer bereits vorhandenen Programmiersprache. Mit Hilfe einer allgemein verwendeten Programmiersprache wird der Tarif abgebildet und anschließend in das Verwaltungsprogramm geladen. Vom Vorteil wäre hierbei die Vereinheitlichung der Programmiersprache des Verwaltungsprogramms und des Tarifprogramms. Eine Interpretation bzw. Kompilation auf Programmebene ist somit nicht mehr nötig. Zentraler Punkt bei dieser Methode ist die Vorgehensweise beim Einbinden des Tarifprogrammes. Je nach verwendeter Programmiersprache kann dieser Schritt anders erfolgen.

Vorteil bei dieser Methodik ist das breite Feld an Möglichkeiten, das aktuelle Programmiersprachen mit sich bringen. Ein weiterer Vorteil ist auch die Möglichkeit der Inkludierung von Bibliotheken, die spezielle Problemstellungen lösen können (z.B. spezielle Mathematik APIs). Dies hat unter anderem bei den bereits angesprochenen Tarifempfehlungen einen großen praktischen Nutzen. Nachteil ist aber, dass die verwendeten Programmiersprachen universelle Programmiersprachen sind und folglich eine geringe Endnutzerfreundlichkeit haben. Bei Anpassungen der Tariflogik muss daher Fachpersonal beansprucht werden, wohingegen bei der *Möglichkeit 1*, in Verbindung mit einer einfachen Syntax, nur unterwiesenes Personal benötigt wird.

Für die entwickelte Programmbibliothek wurde die zweite Methode verwendet, da dies unter anderem der zeitliche Horizont nicht anders erlaubte. Dennoch

hatte diese Entscheidung auch Vorteile. Hinsichtlich der Tarifempfehlungen entstand nämlich eine Notwendigkeit von externen Bibliotheken, die unter anderem das Lösen von Optimierungsproblemen ermöglichten. Somit wird nun im kommenden Abschnitt eine Softwarearchitektur beschrieben, die im Kontext der *Methode 2* entworfen wurde.

Da die entwickelte Programmbibliothek mit Hilfe und für die Programmiersprache Java entwickelt wurde, stellt sich grundsätzlich die Frage, wie man bei dieser Programmiersprache Programmcode dynamisch laden kann. Java ermöglicht mit dem sogenannten `java.lang.ClassLoader` Java Programmcode während der Laufzeit zu laden[23]. Somit wäre bereits ein Teil des Kernproblems K3 gelöst. Denn wenn es unter der Programmiersprache Java zu jeder Zeit möglich ist, neuen Programmcode zu laden, ist es auch zu jeder Zeit Möglich, die Tariflogik zu aktualisieren. Wie nun die nötige Architektur aussieht, wird im nächsten Abschnitt beschrieben. Grundsätzlich müssen aber noch folgende Punkte beachtet werden.

P1. Tarifprogramme müssen eindeutig identifizier- und unterscheidbar sein.

P2. Tarifprogramme müssen selbst definierend sein. Das bedeutet, dass das Tarifprogramm Informationen der Tarifvektoren besitzen muss. Diese Informationen müssen auch zugänglich für das Verwaltungsprogramm sein.

P3. Tarifprogramme müssen stets die gleichen Schnittstellen besitzen damit das Verwaltungsprogramm auf alle Tarife ident zugreifen kann.

P4. Das Testen der Tarifprogramme muss auch ohne ein Aktivsystem möglich sein.

P5. Es muss nachvollziehbar sein wer, das Tarifprogramm geschrieben hat und ob es integer ist.

[23] Nähere Informationen unter Java Platform SE 8 Documentation – java.lang.ClassLoader

Um nun das Kernproblem K3 auf einer hohen Abstraktionsebne lösen zu können, wurde das sogenannte *Factory Methode Pattern*[24] verwendet. Der Einsatz dieses Patterns ermöglicht unter anderem, dass sich die Tarifanwenderseite(z.B. das Verwaltungsprogramm) nicht um das Classloading der Tariflogiken kümmern muss. Umgesetzt wurde dieses Pattern durch die unten angeführten Klassen, die im Package `at.tugraz.ist.javarates` zusammengefasst wurden(grafisch dargestellt in Abbildung 8).

`Rate`: Objekte dieser Klasse repräsentieren einen abgebildeten Tarif. Die nötige Tariflogik wird anhand eines `RateImplementation` Objekt zur Verfügung gestellt.

`RateFactory`: Diese Klasse ermöglicht es, ein konkretes `RateImplementation` Objekt zu erzeugen. Mit Hilfe des Parameters `classPath` kann diese Factory auch nicht geladene Klassen nachträglich einbinden.

`RateImplementation`: In den Subklassen dieser abstrakten Klasse befinden sich die Tariflogikprogramme. Diese Klasse bietet eine einheitliche Struktur und definiert durch abstrakte Methoden, in welcher Methode sich das Tariflogikprogramm befinden muss und wie die Übergabeparameter aussehen müssen.

Durch die oben beschriebene Klasse `Rate` steht dem Verwaltungsprogramm eine einzige Klasse zur Verfügung, dessen Objekte als universelle Tarife angesehen werden können [P3]. Durch die Übergabe der Parameter `className` und `classPath` ist es möglich, im universellen Tarifobjekt die nötige Tariflogik zu setzen. Dieser Schritt wird mit Hilfe der Klasse `RateFactory` durchgeführt.

Durch das Aufrufen der `RateFactory` wird ein konkretes Objekt der Klasse `RateImplementation` erzeugt, welches man über das `Rate` Objekt ansteuern kann. Um nun prosaisch definierte Tariflogiken in Java Code abzubilden, muss

[24] [Gof:107] - Design Patterns: Elements of Reusable Object-Oriented Software

man nur mehr eine Subklasse von `RateImplementation` erzeugen und den Bytecode dieser Klasse im Dateisystem des Verwaltungsprogramms hinterlegen.

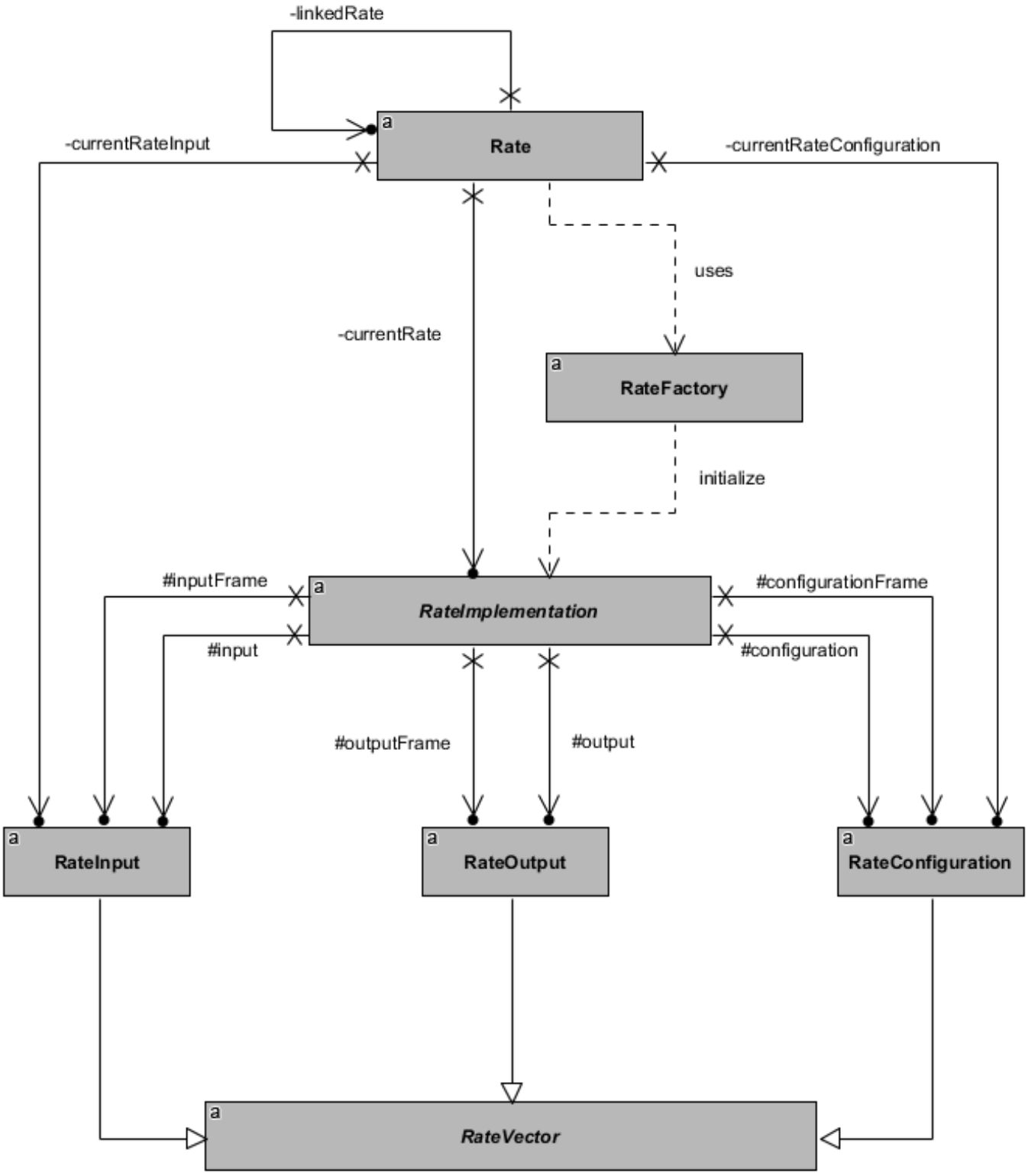


Abbildung 8 JavaRates Architektur

Betrachtet man die abstrakte Klasse `RateImplementation` genauer, so kann man erkennen, dass sie die Attribute `id` und `temporaryId` besitzt. Mit diesen Attributen ist es möglich, die verwendeten Tarife einfach zu identifizieren [P1]. Das ist vor allem nötig, um bei Tarifkompositionen auch auf Zwischenergebnisse zuzugreifen, da die gesamten Tarifausgaben in einem einzigen `RateOutput` Objekt abgelegt werden.

Des Weiteren findet man in der abstrakte Klasse `RateImplementation` eine `initRateEnvironment` Methode. In dieser Methode kann man die `RateImplementation` hinsichtlich ihrer Schnittstellen definieren. Im Allgemeinen bedeutet das, dass man die Zusammenstellung des Eingabe-, Ausgabe- und Konfigurationsvektoren definiert. Somit weiß das Verwaltungsprogramm, wie der passende Eingabe-, Ausgabe- oder Konfigurationsvektor aussehen muss. Hierfür werden die bereits beschriebenen Klassen `RateInput`, `RateOutput` und `RateConfiguration` verwendet [P2].

Die wichtigsten Punkte P1 bis P3 sind somit erfüllt. Der vierte Punkt kann auch sehr schnell abgehandelt werden. Durch das Verwenden der JavaRates Bibliothek ist es jederzeit möglich Tarifprogramme zu entwickeln und sie anschließend auch automatisch zu testen. Durch das Einbinden dieser Bibliothek hat man auch alle nötigen Klassen, um eine konkrete `RateImplementation` zu testen. Problematisch kann es nur werden, wenn man Tarifkompositionen testen möchte. Hierbei müssen natürlich alle nötigen Tarifprogramme in Form von `class` Files zugänglich sein. Eine idente Tarifdatenbank von Produktiv- und Testsystem wäre somit erforderlich [P4].

Der letzte offene Punkt, der noch nicht reflektiert worden ist, beschäftigt sich mit der Authentizität eines Tarifes. Da das Produktivsystem mit Hilfe der JavaRates Bibliothek die Möglichkeit hat jeglichen Java Code auszuführen, ist es sehr wichtig zu wissen, was beim Classloading in das Verwaltungsprogramm geladen wird. Grundsätzlich bestünde die Möglichkeit durch geschickte Angriffe Subklassen von `RateImplementation` auszutauschen. Die vertauschten Klassen mit Schadcode könnten im Falle ihrer Verwendung das Produktivsystem beeinträchtigen bzw. schädigen. Somit ist es sehr wichtig, dass die zur Laufzeit

geladenen `RateImplementation` Klassen hinsichtlich ihrer Authentizität überprüft werden. Diese Überprüfung wird mit Hilfe des `java.net.URLClassLoader` initiiert und erfolgt anhand des `java.lang.SecurityManager`[25]. Der `SecurityManager` überprüft die Signaturen der JAR Dateien, die die `RateImplementation` Klassen beinhalten. Sind diese Signaturen nicht gültig, werden die unpassenden Klassen nicht geladen. Für diesen Schutz wird vorausgesetzt, dass der `SecurityManager` aktiviert ist und dass die nötigen Policies gesetzt sind. Auch ist es nötig, dass die in der Policy Datei verwendeten Zertifikate in einem Java Keystore hinterlegt sind[26]. Da es für Java eine Vielzahl von Systemumgebungen gibt (z.B. J2EE, J2SE, J2ME, u.s.w.) und es auch vorkommen kann, dass das Verwaltungsprogramm zum Beispiel über ein Wirtsystem(Servelt Container) ausgeführt wird, muss die passende Policy Datei gesucht und angepasst werden. Um das Policy Management einfach zu gestalten, sollten daher alle JAR Dateien mit den `RateImplementation` Klassen unter einer Code-Basis (Dateipfad) liegen. Werden alle nötigen Vorkehrungen getroffen, so kann man im Grunde sicher sein, dass das ausgeführte Tarifprogramm authentisch ist und keinen fremden Schadcode beinhaltet [P5].

Alle oben angeführten Probleme konnten nun behandelt und berücksichtig werden. Das Kernproblem K3, als auch die Probleme P1 bis P5 wurden im Entwurf der JavaRates Bibliothek beachtet und ins Design integriert.

Um die praktische Verwendung der `Rate` Klassen zu illustrieren, wird in der Abbildung 9 ein einfaches Anwendungsbeispiel angeführt. Im konkreten Beispiel wird anhand des `Mengen` Tarifs (`com.sampleproject.rates.Amount` – siehe Abbildung 10) das Ansprechen eines `Rate` Objektes dargestellt.

Abschließend sollten auch die internen Abläufe besser beleuchtet werden. Anhand des in der Abbildung 11 beschriebenen Sequenzdiagramms kann man sehr gut mitverfolgen, wie die definierten Klassen untereinander zusammen

[25] Nähere Informationen unter Java Platform SE 8 Documentation – java.net.URLClassLoader

[26] Java Platform SE 7 Documentation – „Java™ Security Overview“ http://docs.oracle.com/javase/7/docs/technotes/guides/security/overview/jsoverview.html

arbeiten. Grundsätzlich kann man den Ablauf in eine Tarifinitialisierungsphase und eine Tarifnutzungsphase einteilen.

```
try {

  Rate rate = new Rate("com.sampleproject.rates.Amount");

  RateConfiguration configuration = rate.getRateConfigurationFrame();
  configuration.addDataIntoField("CONFIG_PricePerUnit", 5.0);

  RateInput input = rate.getRateInputFrame();
  input.addDataIntoField("INPUT_Amount", 5.0);

  rate.setRateConfiguration(configuration);
  rate.setRateInput(input);

  RateOutput output = rate.calculate();

  System.out.println(output.toString());

} catch (RateException ex) {
  ex.printStackTrace();
}
```

Abbildung 9 Verwendung eines JavaRates Tarifes unter Java

```
package com.sampleproject.rates;

import at.tugraz.ist.javarates.*;

public class Amount extends RateImplementation {
  public Amount() throws RateException {
    super();
  }

  @Override
  protected void initRateEnvironment() throws RateException {
    id = "Amount";
    inputFrame.addField(new RateInputField("INPUT_Amount", Double.class, null, "Menge", "", true));
    configurationFrame.addField(new RateConfigurationField("CONFIG_PricePerUnit", Double.class, null, "Stückpreis", ""));
    outputFrame.addField(rateUUID, new RateOutputField("OUTPUT_Price", Double.class, null, "Preis", ""));
  }

  @Override
  protected void solve() throws RateException {

    double amount = (Double) input.getField("INPUT_Amount").getDatum();
    double pricePerUnit = (Double) configuration.getField("CONFIG_PricePerUnit").getDatum();

    double price = amount * pricePerUnit;

    RateOutputField priceField = output.getField(rateUUID, "OUTPUT_Price");
    priceField.setDatum(price);

    output.updateField(rateUUID, priceField);
  }

}
```

Abbildung 10 Stücktarif abgebildet als RateImplementation Klasse

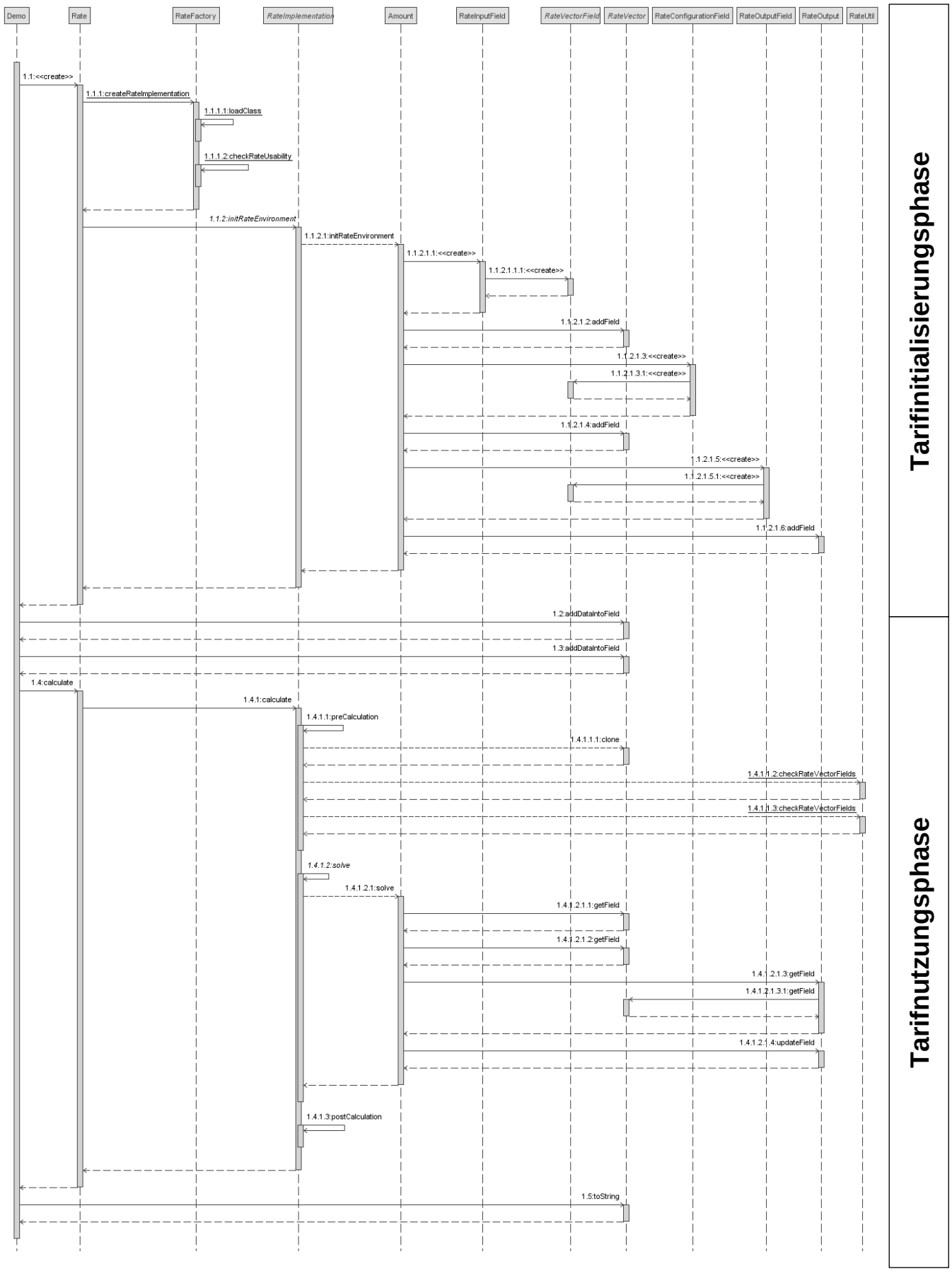


Abbildung 11 UML-Sequenzdiagramm – JavaRates Aufruf

2.3.2.3 *Abbildung von Tarifkompositionen*

Wie bereits im mathematischen Modell beschrieben wurde, ist es auch in der praktischen objektorientierten Umsetzung nötig, dass man Tarife miteinander kombinieren kann. In der JavaRates Bibliothek kann man daher jedem `Rate` Objekt auch ein weiteres `Rate` Objekt zuweisen (`Rate.linkedRate`). Damit können in der Praxis Tarifketten realisiert werden, die kaskadiert berechnet werden können.

So wie es im mathematischen Modell bereits erwähnt wurde, ist es bei manchen Fällen nötig, dass man eine Mappingfunktion einbauen muss, um die Ausgabewerte des vorangegangen Tarifes als Eingabewerte des zu berechnenden Tarifes verwenden zu können. Im Konkreten bedeutet das, dass man aus dem `RateOuput` Objekt, ein `RateInput` Objekt erzeugen muss. Mit Hilfe eines `MappingFunction` Objekts ist es bei Verwendung der JavaRates Bibliothek möglich, dieses Mapping abzubilden. Praktisch heißt das, dass in der Methode `doMapping` der Klasse `MappingFunction` durch Zuhilfenahme der Methodenparameter `linkedRateTemporaryID` und `output` die Vektorfelder des Methodenparameters `input` befüllt werden. Dieses `RateInput` Objekt wird anschließend in der Methode `Rate.calculate()` der Klasse `Rate` als Tarifeingabe zur Berechnung des gegenwertigen Tarifes verwendet.

Wie man eine solche Tarifkomposition erzeugen kann, ist anhand einer Verkettung von zwei `Mengen` Tarifen (`com.sampleproject.rates.Amount`) in der Abbildung 13 ersichtlich. In diesem Beispiel ist auch schön zu sehen, wie anhand der anonymen Klasse `MappingFunction` die Tarifmappingfunktion definiert wird. Zur besseren Verdeutlichung, wird anhand des im Kapitel 2.1.1 definierten Graphenmodels das Beispiel auch noch als Graph illustriert (Abbildung 12).

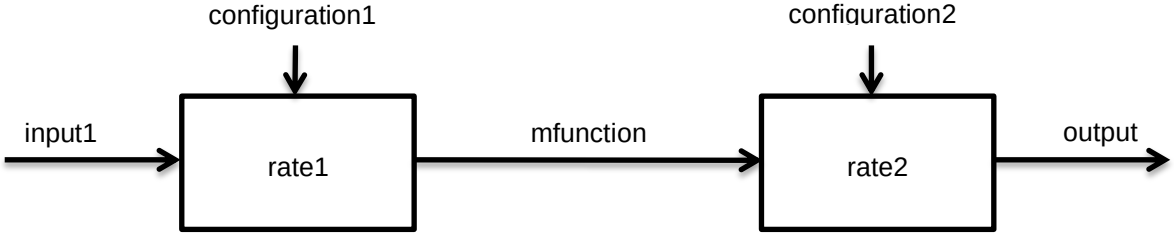


Abbildung 12 Tarifkomposition als gerichteter Graph

```
try {

  Rate rate1 = new Rate("com.sampleproject.rates.Amount");
  Rate rate2 = new Rate("com.sampleproject.rates.Amount");

  RateConfiguration configuration1 = rate1.getRateConfigurationFrame();
  configuration1.addDataIntoField("CONFIG_PricePerUnit", 2.0);

  RateConfiguration configuration2 = rate2.getRateConfigurationFrame();
  configuration2.addDataIntoField("CONFIG_PricePerUnit", 5.0);

  RateInput input1 = rate1.getRateInputFrame();
  input1.addDataIntoField("INPUT_Amount", 5.0);

  rate1.setRateInput(input1);
  rate1.setRateConfiguration(configuration1);
  rate2.setRateConfiguration(configuration2);

  MappingFunction mfunction = new MappingFunction() {
     @Override
     public void doMapping(String linkedRateTemporaryID, RateOutput output,RateInput input) throws RateException {
        input.addDataIntoField("INPUT_Amount",output.getField(linkedRateTemporaryID+".OUTPUT_Price").getDatum());
     }
  };   }

  rate1.linkRate(rate2,mfunction);

  RateOutput output = rate1.calculate();

  System.out.println(output.toString());

} catch (RateException ex) {
  ex.printStackTrace();
}
```

Abbildung 13 Tarifkomposition mit Hilfe von JavaRates

2.3.2.4 *Tarifempfehlung unter JavaRates*

Tarifempfehlungen können, wie es bereits im Kapitel 2.2 beschrieben worden ist, in zwei Anwendungsfälle aufgeteilt werden. Für dieses Kapitel liegt das primäre Interesse am UC2 (siehe Kapitel 2.2.3). Grund für diese Priorisierung ist, dass im Falle vom UC1 (siehe Kapitel 2.2.2), die gewählte Architektur der JavaRates Bibliothek nicht verändert wird. Konkret bedeutet das, dass das eigentliche Vektoroptimierungsverfahren zur Ermittlung des optimalen Tariftupels nicht Teil der JavaRates Bibliothek sein muss, weil das Optimierungsverfahren nur die ermittelten Tarifausgaben benötigt, um das Optimum zu finden.

Hingegen ist beim UC2 eine strukturelle Relevanz hinsichtlich der Architektur der JavaRates Bibliothek vorhanden. Wie bereits in der theoretischen Erörterung der Tarife diskutiert wurde, kann es durchaus vorkommen, dass bei einer Tarifeingabe nicht alle Tarifeingabeelemente vorhanden sind. Folglich ist es notwendig, anhand der vorliegenden Eingaben, eine optimale Lösung zu finden. Die Ermittlung der Lösung muss im Gegensatz zum UC1 in der JavaRates Bibliothek erfolgen bzw. definiert werden.

Betrachtet man nun das Klassendiagramm der JavaRates Bibliothek, so kommen nur bestimmte Klassen in Frage, in denen man diesen Anwendungsfall abbilden kann. Hierbei handelt es sich um die Subklassen der `RateImplementation` Klasse. Die passende Logik für den Fall, dass die Tarifeingabe lückenhaft ist, sollte gekapselt in einer bestimmten Methode der angesprochenen Subklassen definiert werden. In der JavaRates Bibliothek wurde diese Methode abstrakt als `RateImplementation.optimize()` definiert und ist daher von jeder Subklasse von `RateImplementation` zu implementieren. In dieser Methode sollte zunächst die Vollständigkeit der Tarifeingabe ermittelt werden, um im Anschluss die nötige Logik zu wählen, welche, falls vorhanden, das Optimum als `RateOutput` Objekt returniert.

Um alles besser zu veranschaulichen, wird nun ein fiktiver mathematisch definierter Tarif als JavaRates Tarif abgebildet. Die anfallenden Optimierungsfälle werden in der bereits oben erwähnten `RateImplementation.optimize()` Me-

thode behandelt und bieten einen groben Einblick, wie man den beschriebenen UC2 im Zuge der Verwendung von JavaRates abwickeln könnte.

Daher sei nun der Eingabe- und Konfigurationsvektor des angesprochen Tarifes wie folgt definiert:

$$\vec{e} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \qquad \vec{k} = (k_1) \qquad x_1, x_2, k_1 \in \mathbb{R}$$

Des Weiteren sei die Tariflogik wie folgt definiert:

$$t_{\vec{k}}(\vec{e}\,) = ({x_1}^2 + {x_2}^2 - x_1 * x_2 + k_1)$$

Betrachtet man nun die zwei Fälle, bei denen eines der Eingabevektorelemente nicht vorhanden ist, so könnte man den optimalen Wert des fehlenden Elements durch eine einfache Kurvendiskussion bestimmen. Konkret würde das bedeuten, dass der optimale Wert des fehlenden Elements genau dort minimal ist, wo auch die Steigung der Funktion null ist.

$$\frac{t_{\vec{k}}(\vec{e}\,)}{dx_1} = 0 \quad bzw. \quad \frac{t_{\vec{k}}(\vec{e}\,)}{dx_2} = 0$$

Würde nun x_1 das variable Eingabevektorelement sein, so könnte man den optimalen Wert bei gegebenen x_2 wie folgt berechnen.

$$x_1 = \frac{1}{2} * x_2$$

Bei x_2 als variables Eingabevektorelement wäre es daher analog zu vorhin

$$x_2 = \frac{1}{2} * x_1$$

Die Abbildung der oben definierten Tariflogik als Programmcode würde in den Methoden `solve()` (siehe Abbildung 14), sowie `optimize()`(siehe Abbildung 15) als Teil einer konkreten Subklasse von `RateImplementation` erfolgen. Anzumerken ist hierbei auch noch, dass durch das Werfen einer `RateException` in der Methode `optimize()` gewisse Eingabevektorkonstellationen als nichtbehandelt deklariert werden können.

```
@Override
protected void solve() throws RateException {
  double time = (Double) input.getField("INPUT_Time").getDatum();
  double distance = (Double) input.getField("INPUT_Distance").getDatum();
  double basicPrice = (Double) configuraton.getField("CONFIG_BasicPrice").getDatum();

  double price = Math.pow(time,2)+Math.pow(distance,2)-(time)*(distance)+basicPrice;

  RateOutputField priceField = output.getField(rateUUID, "OUTPUT_Price");
  priceField.setDatum(price);
  output.updateField(rateUUID, priceField);
}
```

Abbildung 14 solve-Methode eines fiktiven Tarifes

```
@Override
protected void optimize() throws RateException {
  double basicPrice = (Double) configuration.getField("CONFIG_BasicPrice").getDatum();
  Double time = (Double) input.getField("INPUT_Time").getDatum();
  Double distance = (Double) input.getField("INPUT_Distance").getDatum();

  if(time == null && distance != null){
    time = 1.0/2.0*(distance);
    output.getField(rateUUID, "OUTPUT_OPT_Time").setDatum(time);
  }
  else if(time != null && distance == null){
    distance = 1.0/2.0*(time);
    output.getField(rateUUID, "OUTPUT_OPT_Distance").setDatum(distance);
  }
  else{
    throw new RateException("Wrong Input");
  }

  double price = Math.pow(time,2)+Math.pow(distance,2)-(time)*(distance)+basicPrice;

  RateOutputField priceField = output.getField(rateUUID, "OUTPUT_Price");
  priceField.setDatum(price);

  output.updateField(rateUUID, priceField);
}
```

Abbildung 15 optimize-Methode eines fiktiven Tarife

Im Grunde könnten mit der oben eingeführten Erweiterung der JavaRates Architektur die nötigen Optimierungen des UC2 zum größten Teil behandelt werden. Dennoch wurde ein Teil des formal definierten Models, welcher in gewissen Fällen unerlässlich ist, noch nicht behandelt. Bei diesem Teil handelt es sich um die sogenannten Randbedingungen (siehe Kapitel 2.2.1). Diese optionalen Randbedingungen können durchaus maßgeblich dafür sein, dass man eine Optimierung überhaupt durchführen kann (z.B. bei nichtlinearer Optimierung). Daher wurde in der JavaRates Bibliothek auch die Klasse `RateCondi-`

`tion` eingeführt. Mit Hilfe von Objekten dieser Klasse ist es möglich, einem konkreten `RateImplementation` Objekt die nötigen Randbedingungen für eine Optimierung zu übergeben. Da die `RateCondition` Klasse sehr abstrakt definiert worden ist, ist es sowohl möglich einfache Randwerte (Zahlenwerte) zu übergeben, die zum Beispiel im Anschluss in bereits vordefinierten Randbedingungen eingesetzt werden, als auch spezielle komplexere Randbedingungen, die zum Beispiel als mathematische Formeln übergeben werden. Die Vielfältigkeit der `RateCondition` Klasse begründet sich im gleichen Aufbau, wie man ihn auch bei der `RateVector` Klasse vorfindet. Durch die Übergabe als `java.lang.Object` kann man sowohl einfache Zahlenwerte (`java.lang.Integer`), als auch komplexe mathematische Objekte (z.B. Objekte aus der Bibliothek Commons Math von Apache™[27]) verwenden, um bei Optimierungsverfahren die nötigen Randbedingungen zu bilden.

2.3.2.5 *Tarifkompositionen bei Tarifempfehlungen*

Wie bereits die Überschrift vermuten lässt, beschäftigt sich dieses Kapitel mit Tarifempfehlungen basierend auf Tarifkompositionen. Im Wesentlichen erläutert dieses Kapitel die Einschränkungen, die bei einer Verwendung der JavaRates Bibliothek bei diesem Thema auftreten können. Auch in diesem Kapitel liegt die Priorität beim UC2, da man hinsichtlich des UC1 keine Einschränkungen durch die Verwendung der JavaRates Bibliothek in kaufnehmen muss. Gegenständlich kann man sagen, dass durch die JavaRates Bibliothek eine Tarifempfehlung, wie sie beim UC1 auftritt, vollends unterstützt wird, da das Berechnen von Tarifkompositionen, mit Hilfe der Bibliothek hinsichtlich der im vorigen Kapitel bereits beschriebenen rudimentären Beschränkungen, im Großen und Ganzen vollständig möglich ist. Eine Tarifempfehlung, die im Zuge des UC2 getätigt werden muss, ist aber bei Verwendung der JavaRates Bibliothek im Gegensatz nur eingeschränkt möglich. Den Grund dafür könnte man prosaisch wie folgt beschreiben (plakative Ausgangslage mit einer Tarifkomposition aus nur zwei Tarifen – natürlich erweiterbar).

[27] http://commons.apache.org/proper/commons-math - 08.04.2017

„Im Falle einer Tarifkomposition aus zwei Tarifen und einem Tarifeingabevektor mit variablen Elementen, kann man nur durch die Anwendung eines Optimierungsverfahrens auf Basis der gesamten Tarifkomposition das wahre Optimum ermitteln. Alleine das Ermitteln des optimalen Tarifeingabevektors auf Basis des ersten Tarifes, liefert nicht in allen Fällen das richtige Resultat, da sich die Optimalität nur auf das Zwischenergebnis (Tarifausgabevektor des ersten Tarifs bzw. Tarifeingabevektor des zweiten Tarifs) bezieht.“

Formal könnte man das auch wie folgt beschreiben.

Sei $\vec{e}$ ein Eingabevektor aus der Menge der möglichen Eingabevektoren E . Des Weiteren bestehe dieser Eingabevektor $\vec{e}$ aus variablen Vektorelementen, sodass es eine zugehörige Menge $E_{\vec{e}}^{*} \subseteq E$ gibt, die alle Eingabevektoren beinhaltet, welche aus dem Eingabevektor $\vec{e}$ gebildet werden können. Ebenso sei das anwendbare Optimierungsfahren zur Ermittlung des optimalen Eingabevektors vereinfacht als folgende Funktion dargestellt.

$$OPT\colon (T, E) \rightarrow E^{*}$$

Wenn nun $n \in \mathbb{N} \wedge n \geq 2$ die Länge einer Tarifkomposition ist, dann ist es auch möglich folgende Aussage zu treffen.

$$Es\ existiert\ eine\ Tariftupelmenge\ T\ und\ ein\ unvollständiger\ Eingabevektor\ \vec{e}$$

$$sodass$$

$$t_{\overrightarrow{k_n}}^{(n)}\left(\dots\ t_{\overrightarrow{k_1}}^{(1)}\left(OPT\left(t_{\overrightarrow{k_1}}^{(1)} \circ \dots \circ t_{\overrightarrow{k_n}}^{(n)}, \vec{e}\right)\right)\right) \neq t_{\overrightarrow{k_n}}^{(n)}\left(\dots\ t_{\overrightarrow{k_1}}^{(1)}\left(OPT\left(t_{\overrightarrow{k_1}}^{(1)}, \vec{e}\right)\right)\right)$$

Um diese Aussage zu belegen, wird nun ein Existenzbeweis durchgeführt (Das Beispiel baut auf das Kapitel 2.2.3, sowie dem Kapitel 2.3.2.4 auf).

$$\vec{e} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad \vec{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}, \quad \overrightarrow{k_1} = (k_1), \qquad \overrightarrow{k_2} = (k_2),$$

$$x_1, x_2, k_1, k_2, w_1 \in \mathbb{R}$$

$$0 \leq w_1 \leq 1 \, , w_2 = 1 - w_1$$

Des Weiteren seien die Tariflogiken wie folgt definiert.

$$t^{(1)}_{\overrightarrow{k_1}}(\vec{e}) = \begin{pmatrix} {x_1}^2 + {x_2}^2 - x_1 * x_2 + k_1 \\ x_1 \end{pmatrix}, \quad t^{(2)}_{\overrightarrow{k_2}}(\vec{e}) = ({x_2}^3 + x_1 + k_2)$$

$$t^{(2)}_{\overrightarrow{k_2}}\left(t^{(1)}_{\overrightarrow{k_1}}(\vec{e})\right) = ({x_1}^3 + {x_1}^2 + {x_2}^2 + x_1 * x_2 + k_1 + k_2)$$

$$Beweis\ Teil\ 1 - t^{(n)}_{\overrightarrow{k_n}}\left(\dots t^{(1)}_{\overrightarrow{k_1}}\left(OPT\left(t^{(1)}_{\overrightarrow{k_1}} \circ \dots \circ t^{(n)}_{\overrightarrow{k_n}}, \vec{e}\right)\right)\right):$$

$$\frac{t^{(2)}_{\overrightarrow{k_2}}\left(t^{(1)}_{\overrightarrow{k_1}}(\vec{e})\right)}{dx_1} = 0$$

$$\frac{t^{(2)}_{\overrightarrow{k_2}}\left(t^{(1)}_{\overrightarrow{k_1}}(\vec{e})\right)}{dx_1} = (3{x_1}^2 + 2x_1 - x_2) = 0 \qquad \rightsquigarrow \qquad x_1 = \frac{\pm\sqrt{3x_2 + 1} - 1}{3}$$

$$Beweis\ Teil\ 2 - t^{(n)}_{\overrightarrow{k_n}}\left(\dots t^{(1)}_{\overrightarrow{k_1}}\left(OPT\left(t^{(1)}_{\overrightarrow{k_1}}, \vec{e}\right)\right)\right):$$

$$\frac{g(t_1, \vec{e}, \vec{w})}{dx_1} = 0$$

$$\frac{g(t_1,\vec{e},\vec{w})}{dx_1} = \frac{({x_1}^2 + {x_2}^2 - x_1 * x_2 + k_1) * w_1 + x_1 * w_2}{dx_1} = 2w_1x_1 - w_1x_2 + w_2 = 0$$

$$x_1 = \frac{1}{2w_1} * (w_1x_2 - w_2)$$

$Beweis\ Teil\ 3 - Einsetzen\ eines\ konkreten\ Eingabevektors$:

$$x_2 = 2 \rightsquigarrow \vec{e} = \begin{pmatrix} x_1 \\ 2 \end{pmatrix}$$

$$\vec{w} = \begin{pmatrix} 0{,}5 \\ 0{,}5 \end{pmatrix}$$

$$t_{\overrightarrow{k_n}}^{(n)}\left(\dots\ t_{\overrightarrow{k_1}}^{(1)}\left(OPT\left(t_{\overrightarrow{k_1}}^{(1)} \circ \dots \circ t_{\overrightarrow{k_n}}^{(n)}, \vec{e}\right)\right)\right) \neq t_{\overrightarrow{k_n}}^{(n)}\left(\dots\ t_{\overrightarrow{k_1}}^{(1)}\left(OPT\left(t_{\overrightarrow{k_1}}^{(1)}, \vec{e}\right)\right)\right)$$

$$\left(\frac{\sqrt{7}-1}{3}\right)^3 + \left(\frac{\sqrt{7}-1}{3}\right)^2 + 2^2 + \left(\frac{\sqrt{7}-1}{3}\right) * 2 \neq \frac{1^3}{2} + \frac{1^2}{2} + 2^2 + \frac{1}{2} * 2$$

$$\sim 4.8837 \neq 5.375$$

Die zu beweisende Aussage ist somit erfüllt.

Die oben getätigte Aussage mit anschließendem Beweis schränkt die Möglichkeiten des Tarifempfehlungsverfahrens beim UC2 enorm ein. Im Konkreten bedeutet das, dass man für die Ermittlung des optimalen Tarifeingabevektors die gesamte Tarifkomposition verwenden muss. Aus mathematischer Sicht ist diese Voraussetzung kein Problem, da die variablen Elemente einer Tarifeingabe einfach mit den Termen des Ausgabevektors der vorangegangenen Tarife ersetzt werden können(siehe Beweis). Als mögliche Folge dieses Lösungsansatzes muss man aber die Eventualität anführen, dass der letzte Ausgabevektor in Folge sehr komplexe Terme enthalten könnte. Betrachte man dieses Problem aber seitens der Informatik, entwickelt sich diese Voraussetzung zu einem großen Problem. Der termbasierte Ansatz aus der Mathematik lässt sich nur

schwer mit Hilfe von einem einfachen Programmcode abbilden. Zum Lösen dieser Problematik würden sich aber dennoch zwei Weg anbieten:

L1. Bei diesem Lösungsansatz würde man das termbasierte Verfahren aus der Mathematik simulieren. Unter Zuhilfenahme einer umfassenden Mathematik Bibliothek (z.B. Commons Math von Apache™) würden man bei Verwendung der JavaRates Bibliothek komplexe Termobjekte (z.B. `org.apache.commons.math4.analysis.UnivariateFunction`) anhand von `RateVectorField` Objekten übergeben. Im Zuge der Optimierung müsste man den resultierenden Term bei jedem Schritt weiterentwickeln. Die Optimierung würde abschließend ebenfalls durch die besagte Mathematik Bibliothek erfolgen. Dieser Ansatz kann als überaus komplex bezeichnet werden und wurde infolge auch nicht weiterverfolgt.

L2. Dieser Lösungsansatz ist im Vergleich nicht so elegant, wie der L1. Dennoch punktet er mit einer gewissen Unkompliziertheit. Im Wesentlichen baut dieser Lösungsansatz auf die Beobachtung auf, dass Tarifkompositionen in der Praxis ehr selten auftreten. Eine Analyse der Tarifbasis hinsichtlich möglicher Tarifkompositionen, könnte man als Ausgangspunkt für die Konstruktion von gekapselten Tarifen auf Grundlage von Tarifkompositionen verwenden. Natürlich ist dieser Schritt nur in den Fällen nötig, bei denen auch eine Tarifoptimierung à la UC2 zu erwarten ist.

Die oben angeführten Lösungsansätze könnten unterschiedlicher nicht sein, dennoch können beide Möglichkeiten durch die JavaRates Bibliothek abgebildet werden.

2.3.3 Zusammenfassung

In diesem Kapitel konnte nun gezeigt werden, wie das hoch abstrahierte Tarifmodel auch als Programmbibliothek anwendbar gemacht werden kann. Anfänglich wurde durch das Ausarbeiten der Schlüsselprobleme versucht, die Modelabbildung auch als Programmcode in einem maximalen Abstraktionslevel abzubilden. In der daraus entstandenen JavaRates Bibliothek konnten infolge alle nötigen Faktoren berücksichtigt werden, welche auch als Kernprobleme K

und als sekundäre Probleme P definiert worden sind. Zur herkömmlichen Tarifberechnung konnte auch die Tarifempfehlung integriert werden und durch die zugrundeliegende Architektur wurden beide Bereiche einer Standardisierung unterworfen. Durch diese Standardisierung wird es nun ermöglicht, dass Tarife stets über die gleiche Schnittstelle nutzbar sind, und andererseits, dass Tarife in einer wohl definierten Umgebung abgebildet und entwickelt werden können.

3. Java EE Framework mit Tarifintegration

In diesem Kapitel wird nun abschließend auf das mehrfach angesprochene konzeptionelle Programmframework eingegangen, welches im Zuge der Diplomarbeit entwickelt wurde. Anfangs wird erörtert, um was es sich bei diesem Framework handelt und welche Ausgangsbedingungen vorhanden waren. Anschließend werden alle getätigten Designentscheidungen beschrieben, die anhand der vorab definierten Randbedingungen getroffen worden sind. Schlussendlich wird noch auf die resultierende Architektur eingegangen und beschrieben, wie die im Kapitel 2.3 entworfene JavaRates Bibliothek in die entwickelte Architektur integriert worden ist.

3.1 Randbedingungen

Beim entwickelten konzeptionellen Programmframework handelt es sich im Wesentlichen um ein Grundgerüst für die Entwicklung von Softwaresystemen im Bereich des Verrechnungsmanagements. Die Beschränkung dieser Arbeit auf ein Framework, basiert auf der eingangs erworbenen Erkenntnis (siehe Kapitel 2 S.19), dass Softwarelösungen für Verrechnungsprozesse einer großen Vielfalt unterliegen. Folglich orientiert sich diese Arbeit auf die Entwicklung einer fundierten Basis, um eine maximale Breite hinsichtlich möglicher abzubildender Verrechnungsprozesse zu erlangen.

Die oben getroffene Randbedingung, dass das Framework als Grundlage für eine breite Masse an Verrechnungsprozessen dienen soll, spricht für eine Integration der JavaRates Bibliothek in das Programmframework. Daher liegt ein weiteres Hauptaugenmerk dieses Abschnittes in der Demonstration einer praktischen Umsetzung der im Kapitel 2 diskutierten Tarifproblematik. Durch die JavaRates Bibliothek kann demgemäß ein solider Grundstein gesetzt werden, da diese, wie bereits gezeigt wurde, auf einem sehr hohen Abstraktionslevel implementiert worden ist.

Weitere grundsätzliche Randbedingungen, welche durch das Framework abgedeckt sein sollten, sind unter anderem eine Kunden-, Produkt- bzw. Leistungs- und Zugangsverwaltung. Da auch diese Bereiche sehr allgemein gehalten sein

sollten, ist speziell darauf zu achten, welche Technologien bzw. Architekturen zum Einsatz kommen. Im nächsten Kapitel werden nun die getroffenen Designentscheidungen beschrieben und die nötigen Technologien angesprochen.

3.2 Designentscheidungen

Dieses Kapitel beschäftigt sich mit Konzepten, die die oben definierten Randbedingungen bestmöglich in das Programmframework einbinden. Anfänglich wird daher eine grundsätzliche Architektur eingeführt und beschrieben, warum sich diese für die Umsetzung optimal eignet. Aufbauend auf diese Entscheidung wird eine Technologie beschrieben, die wiederum helfen sollte, diese Architektur bestmöglich abzubilden. Abschließend werden mithilfe von Teiltechnologien der vorweg eingeführten Technologie, Designpatterns illustriert, welche wiederum zu einer Erhöhung des Abstraktionslevels beitragen sollen.

3.2.1 Client/Server Architektur

Eine grundlegende Entscheidung bei der Entwicklung einer Softwarelösung ist die Wahl, wie und wo eine Software ausgeführt werden sollte. Demgemäß unterscheidet man verteilte bzw. nicht verteilte Softwaresysteme. Der wesentliche Unterschied der beiden Systeme ist der Ort, an dem die Software ausgeführt wird **[14]**. Beim nicht verteilten Ansatz wird die komplette Softwarelösung auf einem einzigen Wirtsystem (Hardware) ausgeführt. Dieser Ansatz scheint alt hergebracht zu sein, dennoch kann man den Sicherheitsaspekt dieser Lösung nicht verschweigen. Da man für diesen Ansatz kein Netzwerk (z.B. Internet) benötigt, muss man das System auch nicht gegenüber Angriffe aus solchen Netzwerken schützen. Dieser Sicherheitsaspekt spielt hinsichtlich der möglichen Kunden- und Finanzdaten, die im Verrechnungssystem verarbeitet werden, durchaus eine große Rolle und sollte nicht außer Acht gelassen werden. Dennoch haben sich verteilte Systeme durchgesetzt. Im Konkreten würde man hierbei von Client/Server Systemen sprechen. Diese Systeme teilen den in der Software abgebildeten Verrechnungsprozess auf mehrere Wirtsysteme auf. Durch die Omnipräsenz des Internets und die steigende Leistungsfähigkeit von lokalen Netzwerken, ist ein stetiger Zuwachs solcher verteilten Systeme wohl

weiterhin zu erwarten. Die Sicherheitsaspekte sind aber infolge der Nutzung von Netzwerken nicht zu unterschätzen und sollten stets hinterfragt werden. Angriffe auf Kundendaten, einschließlich Kreditkarteninformation, sind nun keine Seltenheit mehr**[15]**. Angesichts dieser Tatsachen wurde das Programmframework aber dennoch als Client/Server System konzipiert, da ein nicht verteiltes System in der heutigen Waren- bzw. Dienstleistungswirtschaft, kaum ein wirtschaftliches Potential hätte.

3.2.2 Java Enterprise Edition

Da nun definiert wurde, welche grundsätzliche Architektur modelliert wird, ist es auch nötig zu definieren, mit welchen Technologien man diese Architektur abbilden möchte. Im Zuge diese Arbeit liegt der Focus primär bei den Softwareaspekten eines verteilten Systems. Daher wird in diesem Abschnitt nicht auf Hardware- und Netzwerktechnologie eingegangen. Für das Abbilden von verteilten Systemen auf Softwareebene, benötigt man daher grundsätzlich auch eine Programmiersprache, die dieses Konzept unterstützt. Die zutreffende Programmiersprache sollte aber auch die Voraussetzungen der JavaRates Bibliothek erfüllen. Daraus resultiert unter anderem auch, dass es keinen Mangel an versiertem Fachpersonal geben darf, welches sich beim Entwickeln mit Hochsprachen auskennt und welches auch mit der gewählten Lösung von JavaRates, Softwaretarife erstellen und supporten kann. Daher sollte die genutzte Programmiersprachenumgebung auch einer breiten Öffentlichkeit zugänglich sein. Betrachtet man gängige Programmiersprachen Rankings, so findet man in vielen dieser Statistiken, Java als abgeschlagene Nummer eins. Diese Gegebenheit und auch die Voraussetzung, dass Java alle nötigen Voraussetzungen für verteilte Systeme erfüllt, sprechen für eine Benutzung dieser Programmierumgebung.

In Folge der Wahl von Java als Programmierumgebung, stellt sich nun auch die Frage, welche Java Plattform genutzt werden soll. Grundsätzlich bietet die Programmiersprache Java vier Plattformen an. Diese wären zunächst Java Card für Chipkarten, Java ME für Embedded Systems, Java SE und Java EE als Plattform für PCs, Server und dergleichen. Da das Framework primär für PC

und Server gedacht sein sollte, bietet sich die Java SE - und Java EE Plattform an. Mit dem monetären Hintergrund, den das Framework besitzt, ist es auch wichtig ein transaktionsbasierte Umgebung zu besitzen, um die Datenbasis stetig konsistent halten zu können. Diese Voraussetzung, sowie auch die Unterstützung des Layer-Patterns und MVC-Patterns, bilden unter anderem die Grundlage für die Wahl der Java Enterprise Edition Plattform. Im nächsten Kapitel wird nun beschrieben, wie mit Hilfe der Java Enterprise Edition das Framework als verteiltes System aufgebaut werden kann. Im Zuge dessen, werden auch alle nötigen Java EE Technologien angeführt, die für eine Umsetzung benötigt werden.

3.2.3 JPA, EJB, JSF oder auch MVC – Pattern

Betrachtet man die Spezifikation der im vorigen Kapitel gewählten Java EE Plattform näher, so wird man feststellen, dass sich diese Java Plattform in Grunde nur durch wenige Technologien bzw. Funktionen von der Java SE (Standard Edition) unterscheidet. Diese fehlenden Funktionen geben aber der Java EE Plattform die zusätzliche Leistungsfähigkeit, die benötigt wird, um verteilte Systeme bestmöglich zu implementieren. Wie bereits vorab definiert wurde, basiert das hier entwickelte Framework auf einer Client/Server Architektur. Um diese Architektur mit Hilfe der Java EE Plattform abzubilden, müssen sowohl auf der Client-, als auch der Serverseite Grundvoraussetzungen geschaffen werden.

Zu den Grundvoraussetzungen auf der Clientseite gehört zum Beispiel ein Webbrowser als Möglichkeit einer universellen Benutzerschnittstelle, oder eine funktionsfähige Java SE Installation mit passenden APIs, als Basis für eine speziell entwickelte Benutzerschnittstelle. Serverseitig ist es zudem noch notwendig, einen passenden Java EE Server (z.B. GlassFish, WildFly etc.) bereit zu stellen, der durch standardisierte Java Objekte (z.B. Enterprise Java Beans (EJB), Java Server Faces (JSF), …), verwaltet durch einen Web Container bzw. EJB Container, Anfragen des Clients bearbeiten kann. Um abschließend auch noch anfallende Daten speichern zu können, empfiehlt sich die Nutzung einer Datenbank. Anhand der Abbildung 16 sind alle oben genannten Technologien

angeführt und ihre Zusammenhänge illustriert. Mit Hilfe der dieser veranschaulichten Architektur kann man sehr einfach modulare verteilte Programme entwickeln.

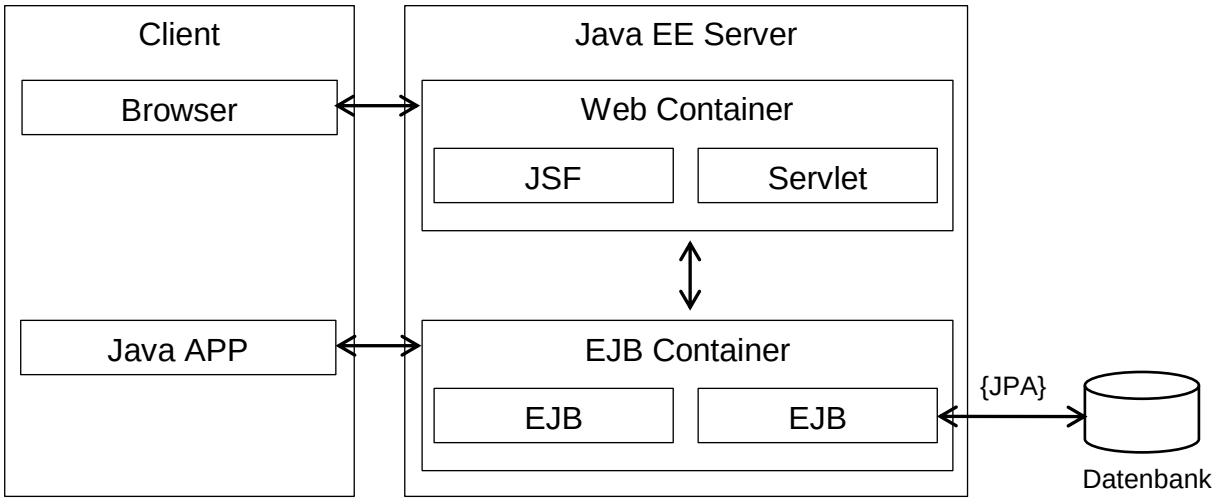


Abbildung 16 Java EE Architektur[28]

Im Grunde entspricht die Architektur der Java EE Plattform einem reduzierten MVC Pattern[29]. Die Verschlankung des MVC Patterns kommt durch das Fehlen des Observer Patterns zustande (zwischen View und Model). Dennoch kann man die angesprochenen Komponenten der Java EE Architektur wie folgt in Relation setzen:

Datenbank	≡	Model
Client (Browser,Java APP)	≡	View
Java EE Server	≡	Controller

Das zweite Design Pattern, das man im Aufbau der Java EE Architektur finden kann, ist das sogenannte Layer Pattern[30]. Demgemäß kann jedes der oben angeführten Komponenten auch als ein Layer gedeutet werden (siehe Abbildung 17).

[28] Java Platform, Enterprise Edition: The Java EE Tutorial - Figure 1-6

[29] [POSA:125] – Model-View-Controller, Pattern-Oriented Software Architecture

[30] [POSA:31] – Layer, Pattern-Oriented Software Architecture

Anhand des Layer Patterns können unter anderem auch gewisse Layer ausgetauscht werden, um in Folge die Funktionalität des Frameworks zu adaptieren. Eine Verfeinerung der Layerstruktur ist durch das angeführte Pattern auch möglich und hilft dabei das Framework modularer aufzubauen. Diese Modularität ist der Kernpunkt, der das Framework auch für ein breites Feld an Verrechnungsprozessen zugänglich macht. Wie nun dieses Modularität genau zustande gebracht worden ist und wie die endgültige Architektur des Frameworks aussieht, ist Thema des nächsten Kapitels.

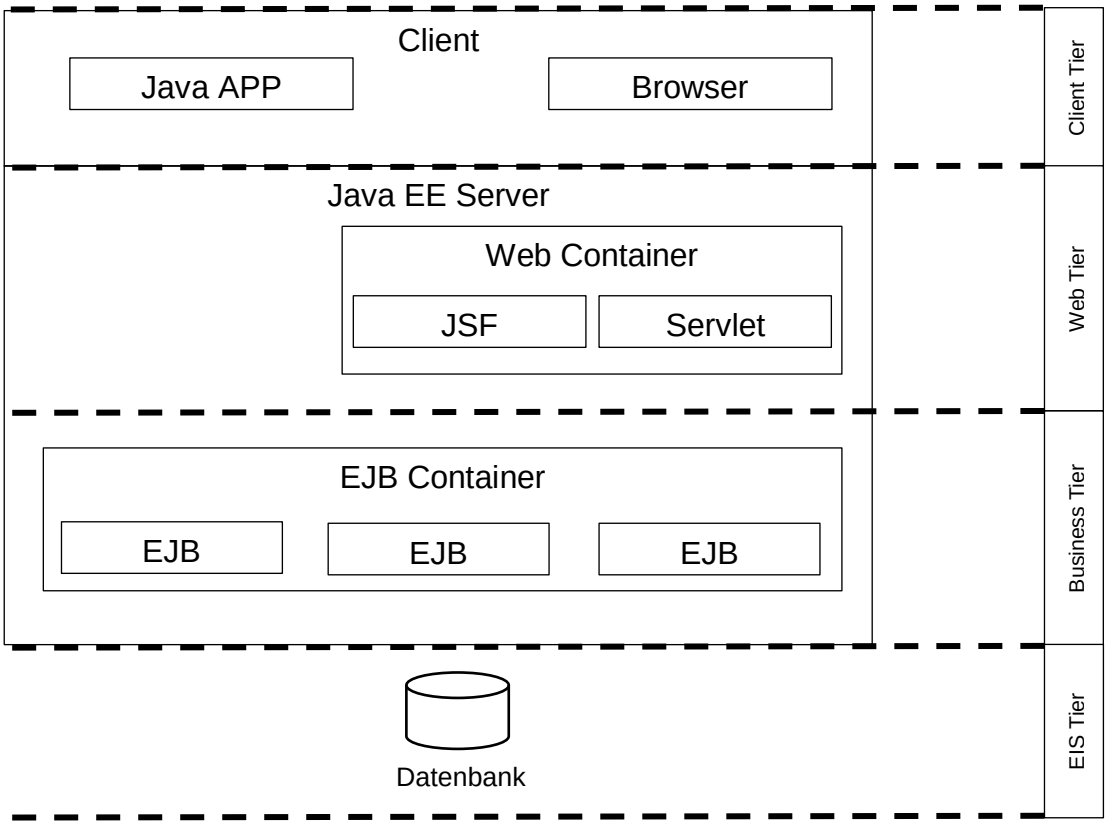


Abbildung 17 Java EE Architektur als Layermodel[31]

3.3 Frameworkarchitektur

In diesem Kapitel wird nun illustriert, wie das konzeptionelle Framework aufgebaut ist. Da es umfangsmäßig nicht möglich ist, auf alle Einzelheiten einzugehen, werden in diesem Kapitel nur jene Teile dargestellt, die dabei helfen einen

[31] Java Platform, Enterprise Edition: The Java EE Tutorial - Figure 1-1

groben Einblick in das Framework zu erlangen. Dennoch sollte anhand dieses Kapitels aber vermittelt werden, wie es möglich ist, die angesprochene Vielfalt an Verrechnungsprozessen abzubilden. Abschließend wird noch demonstriert, wie die JavaRates Bibliothek in das Framework eingebunden wurde.

3.3.1 Gesamt Architektur

Auf Basis der bereits besprochenen Designentscheidungen wurde eine Framework Architektur entwickelt, die als Grundlage für ein breites Feld an Verrechnungsprozessen dienen soll. Um nun diese Voraussetzung zu erfüllen, wurden mehrere Designpatterns integriert. Betrachtet man nun die entworfene Architektur (siehe Abbildung 19), so kann man die angesprochenen Patterns sehr gut wiederfinden. Sowohl das Konzept von verteilten Systemen, als auch das MVC Pattern, sind auf den ersten Blick leicht erkennbar. Des Weiteren kann das Layer Pattern nach genauer Betrachtung auch noch gefunden werden. Demgemäß kann man zum Beispiel erkennen, dass das Modul „Kundenverwaltung“ auf dem „Basis“ Modul aufbauend ist. Da nun auch von Modulen die Rede ist, ist es nötig, auf eine bereits angesprochene Technologie der Java EE Plattform erneut einzugehen und die Notwendigkeit dieser Technologie für das Framework zu beschreiben.

Im vorherigen Kapitel wurden die s.g. Enterprise Java Beans (EJB) angesprochen. Diese Beans sollten laut dem Java EE Konzept die Businesslogik beinhalten und sich in der Controller-Schicht befinden. EJB werden durch den EJB Container eines Java EE Servers verwaltet und bereitgestellt. Ein Teil der Arbeit von EJBs ist auch die Bereitstellung der Model-Schicht. Benötigt zum Beispiel die View-Schicht Zugriff auf das Model, so greift sie auf das passende Bean der Controller-Schicht zu. Durch die Verwendung von Transaktionen[32] (Container-Managed Transactions, Bean-Managed Transactions) ist es auch möglich, ein hohes Maß an Persistenz in der Model-Schicht zu erreichen. Eine weitere Besonderheit dieser Technologie ist die Art und Weise, wie man ein konkretes

[32] Java Platform, Enterprise Edition: The Java EE Tutorial - Transactions

EJB einbinden kann. Durch eine s.g. „Dependency Injection“ oder einem „JNDI lookup“[33] ist ein Zugriff sowohl lokal als auch remote möglich.

Betrachtet man nun die entwickelte Framework Architektur, so bildet genau die Dependency Injection (DI) den Hauptgrund für das hohe Maß an Erweiterbarkeit. Unter Zuhilfenahme der DI ist es sehr einfach möglich, dass die vorhandenen Module und ihre Funktionalität im gesamten Framework nutzbar sind. So kann zum Beispiel das „Rechnungslegung /-verwaltung“ Modul auf Kundendaten zugreifen, ohne selbst Wissen über das Model des „Kundeverwaltung“ Moduls zu haben. Durch Hinzufügen von weiteren Modulen (in der Abbildung 19 angedeutet als Modul „XY“) kann man nun den gewünschten Verrechnungsprozess einbinden, und falls notwendig, Funktionen von vorhandenen Modulen einbinden.

Das gewünschte Ziel, ein Framework zu entwickeln, sollte genau durch die oben beschriebene Modularität zu einem hohen Maß erreicht worden sein. Im anschließenden Kapitel wird zudem noch durch die Integration der JavaRates Bibliothek in das Framework eine zusätzliche Abstraktion erreicht, und dadurch auch die Nutzungsweite des Frameworks zusätzlich vergrößert.

3.3.2 JavaRates Integration

Die in dieser Arbeit entwickelte JavaRates Bibliothek wurde sehr abstrakt konzipiert. Der Grund dafür ist, dass nur durch ein abstraktes Design die Möglichkeit geschaffen wurde, dass die Bibliothek in vielen unterschiedlichen Verrechnungsprozessen wiederverwendet werden kann. In der Bibliothek wurde daher vieles nicht implementiert, was man im Zuge einer praktischen Anwendung vielleicht benötigen könnte. Dennoch wurde aber versucht, dass die Implementation weitestgehend erweiterbar ist, um alle nötigen erweiternden Funktionen einbinden zu können.

[33] Java Platform, Enterprise Edition: The Java EE Tutorial - Accessing Enterprise Beans

Bei der Entwicklung des Frameworks konnten nun folgende drei Probleme ermittelt werden, welche nicht bzw. nur teilweise durch die JavaRates Bibliothek abgedeckt werden.

P1. Beim ersten Problem geht es grundsätzlich darum, dass eine Tarifkonfiguration, Tarifeingabe und Tarifausgabe speicherbar sein muss. Die in der Bibliothek verwendeten Objekte, die dafür zuständig sind, dass die Tarifvektoren abgebildet werden, können als einfache Java-Objekte nur schwer permanent abgespeichert werden. Somit ist es nötig, diese Objekte in eine speicherbare Form zu transformieren.

Lösung: Bei diesem Problem bietet sich die Speicherung der Vektoren in einem XML Format sehr gut an. Die hohe Flexibilität von XML wird auch benötigt, um das hohe Maß an Abstraktheit der JavaRates Bibliothek weiterhin gewährleisten zu können. Um das Marshalling einfacher zu gestalten, wurde die Java Bibliothek, JAXB[34] eingesetzt. Die statischen Methoden der Klasse `at.tugraz.ist.emsfjavarates.xml.XMLVectorPrinter` liefern XML Repräsentationen von Objekten der Klassen `RateInput`, `RateOutput` und `RateConfiguration`. Die durch die Methoden erstellten `String` Rückgabewerte sind in weiterer Folge einfacher zu hinterlegen, als die Java-Objekte (siehe Abbildung 18).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xmlVector>
    <fields>
        <entry>
            <datum xsi:type="xs:double" xmlns:xs="…" xmlns:xsi="…">5.0</datum>
            <description>Menge in Ganzen Zahlen</description>
            <id>INPUT_Amount_1</id>
            <name>Menge</name>
            <requiert>true</requiert>
            <type>java.lang.Double</type>
            <unit></unit>
        </entry>
    </fields>
</xmlVector>
```

Abbildung 18 RateVector Objekt repräsentiert als XML String

[34] JAXB - Java Architecture for XML Binding - https://docs.oracle.com/javase/tutorial/jaxb

P2. Da man bei der Nutzung des Frameworks auch mit der JavaRates Bibliothek interagieren muss, ist es hinsichtlich der Benutzerfreundlichkeit sehr wichtig, dass die nötigen Objekte, mit denen kommuniziert werden muss (Tarifeingabe,-ausgabe und -konfiguration), auch alle nötigen Informationen enthalten, die zu einer besseren Eingabefreundlichkeit verhelfen. Betrachtet man grundsätzlich die `RateVector` Klasse der JavaRates Bibliothek, so findet man in ihr keine zusätzlichen Informationen über eine Reihenfolge der `RateVectorField` Eingaben. Im Hinblick der Benutzerfreundlichkeit ist es aber sinnvoller, eine gewisse Eingabestruktur zu definieren, um mögliche Verwirrungen vorzubeugen.

Lösung: Um das oben beschriebene Problem zu lösen, wurden Subklassen der Klassen `RateInputField`, `RateOutputField` und `RateConfigurationField` definiert. Diese Klassen besitzen ein zusätzliches Attribut, welches als Strukturinformation fungiert.

P3. Beim dritten Problem handelt es sich ebenfalls um ein Problem der Benutzerfreundlichkeit. Im Konkreten kann es bei manchen Fällen vorkommen, dass es bei gewissen Tarifeingaben nur eine diskrete Menge an Eingabewerten gibt bzw. geben soll. Die konkrete Menge der möglichen Eingabewerte sollte bestenfalls bereits bei der Eingabe bekannt sein und nicht erst als Fehlerrückgabe geliefert werden. An sich könnte man dieses Verhalten auch ohne Erweiterungen bereits durch eigens definierte Klassen abdecken. Jedoch wäre dieser Weg sehr umständlich.

Beispiel: Bei einem beliebigen Tarif sind als Tarifeingabe nur die `Integer` Werte 5,10 und 15 zulässig. Bei der angesprochenen umständlichen Lösung würde man eine Subklasse von `Integer` ableiten, welchen nur die gewünschten Werte annehmen kann. Bei der Benutzeroberfläche würde man infolge anhand der definierten Klasse eine passende Tarifeingabe modellieren können.

Lösung: Ein besserer Lösungsweg wäre das Erweitern der Lösungsmethode von P2. Für das Lösen dieses Problems müsste man ein weiteres Attribut hinzufügen, welches alle nötigen Tarifeingaben definieren kann. Eine

Implementierung des `java.util.Collection` Interfaces könnte eine solche Datenstruktur sein, in der diese Werte hinterlegt werden können.

Die oben angeführten Probleme P1 bis P3 wurden im Package `at.tugraz.ist.emsfjavarates` zusammengefast und sind Teil des konzeptionellen Frameworks. Im Zuge des Integrierens der EMSFJavaRates Bibliothek in das entwickelte Framework, musste aber noch die passende Businesslogik, Persistenz-Schicht, sowie Benutzeroberfläche des beteiligten Moduls entwickelt werden. Im Folgenden werden nun kurz die wichtigsten Punkte beschrieben, die im Zuge der Integration zu beachten waren.

Wie man aus der Abbildung 19 erkennen kann, ist die EMSFJavaRates Bibliothek ein Teil des „Tarifverwaltung“ Moduls. Die grundsätzliche Funktion dieses Moduls ist die Bereitstellung und die Verwaltung von Tarifen. Unter dem Bereich der Bereitstellung fällt im Grunde die Entwicklung der Benutzeroberfläche für die Interaktion zwischen Tarif und Benutzer, sowie die Implementierung der EMSFJavaRates Schnittstelle. In den Bereich der Verwaltung fällt das Entwickeln der Verwaltungsoberfläche, aber auch das Konstruieren der Persistenz-Schicht, damit Tarife hinterlegt, konfiguriert und erneuert werden können.

Verwaltung von Tarifen: Da jeder Tariftupel modellgemäß aus zwei Teilen besteht (Tariflogik, Tarifkonfiguration), müssen auch beide Elemente zugänglich gemacht werden. Durch die Verwendung der JPA[35] Technologie im Bereich der Persistenz-Schicht, sollten beide Komponenten durch Objekte abgebildet werden.

Im Falle der Tarifkonfiguration ist ein einfacher `String` ausreichend, da man mit Hilfe der EMSFJavaRates Bibliothek, `RateVector` Objekte als XML formatierte `String` Objekte darstellen kann. Im Falle der Tariflogik ist dies nicht so einfach. Da es sich bei der Tariflogik um eine `class`-Datei (also Java Bytecode) handelt und diese Datei wegen dem Sicherheitsaspekt (siehe dazu Kapitel 2.3.2.2 Problem P5) in einem signierten JAR Archiv hinterlegt

[35] Java Platform, Enterprise Edition: The Java EE Tutorial - Persistence

werden muss, ist es sinnvoller, die Tariflogik direkt auf dem Filesystem des Wirtbetriebssystems zu hinterlegen. Der Ordner, in dem die JAR-Archive gespeichert sind, sollte infolge auch in der Policy Datei des Java EE Servers vermerkt sein. Auch müssen die Zertifikatsschlüssel für die Integritätskontrolle im Zertifikatsspeicher des Java Runtime Environment hinterlegt sein. Die Information, wo nun das richtige JAR-Archiv zu finden ist, kann aber in Folge anhand eines einfachen `String` Objekt zugänglich gemacht werden.

Zusammenfassend wurde daher eine JPA Entity Klasse definiert, welche beide Informationen (XML formatierter Tarifkonfiguration, Systempfad zum JAR-Archiv) abbildet.

Bereitstellung von Tarifen: Bei der Bereitstellung der Tarife befand sich der primäre Focus in der View-Schicht. Da die Benutzeroberfläche mit Hilfe der JSF[36] Technologie entwickelt worden ist, und die Kommunikation zum Client auf HTML, JS, und CSS basiert, konnte man für die Tarifeingabe auf eine Summe an Eingabeelementen zurückgreifen. Um die Benutzerfreundlichkeit noch zu erweitern, wurde zusätzlich noch ein auf JSF aufgebautes User Interface Framework verwendet. Das s.g. PrimeFaces[37] Framework bietet zu den Standardeingabeelementen weitere sehr nützliche Bedienelemente, die eine Nutzung der Tarife um ein Vielfaches erleichtern.

Anhand des Tarifeingabeframes, der am Beginn einer Tarifnutzung abfragbar ist, wird mit Hilfe der s.g. DynaForms[38] aus der API „PrimeFaces Extensions“ ein Eingabeformular erstellt. Die passenden Tarifeingabeelemente werden anhand des Attributes `Typ` der abzufragenden `RateInputField` Objekte ermittelt. Standard Typen wie `Integer`, `String`, `Double`, etc. können vorab im Modul definiert sein. Komplexere Eingabeelemente (z.B. GoogleMaps-Koordinaten für die Ermittlung einer Wegstrecke) können als Teil jenes JAR-

[36] Java Platform, Enterprise Edition: The Java EE Tutorial - JavaServer Faces Technology
[37] https://www.primefaces.org
[38] https://www.primefaces.org/showcase-ext/views/home.jsf - DynaForm - 13.07.2017

Archiv bereitgestellt werden, welches auch die `class`-Datei der Tariflogik hinterlegt hat.

Der oben angeführte Ansatz zur Integration der JavaRates Bibliothek wurde im Modul „Tarifverwaltung“ implementiert und steht als Java Package `at.tugraz.ist.emsf.rates` zur Verfügung. Der Aufbau des Packages ist angelehnt am Layer-Pattern. Daher gibt es auch folgende Subpackages mit passenden Korrelationen.

`at.tugraz.ist.emsf.rates.service` – Businesslogik-Schicht
`at.tugraz.ist.emsf.rates.model` – Persistenz-Schicht
`at.tugraz.ist.emsf.rates.controller` – View-Schicht

Das Package der View-Schicht wird noch durch zusätzliche xHTML-Dateien in einem separaten Ordner erweitert.

Mit den oben angeführten Informationen, welche beschreiben, wie nun die JavaRates Bibliothek als Modul des konzeptionellen Frameworks integriert worden ist, sowie einem kurzen Code-Studiums, sollte es sehr schnell möglich sein, weitere Module zu entwickeln und bereitzustellen.

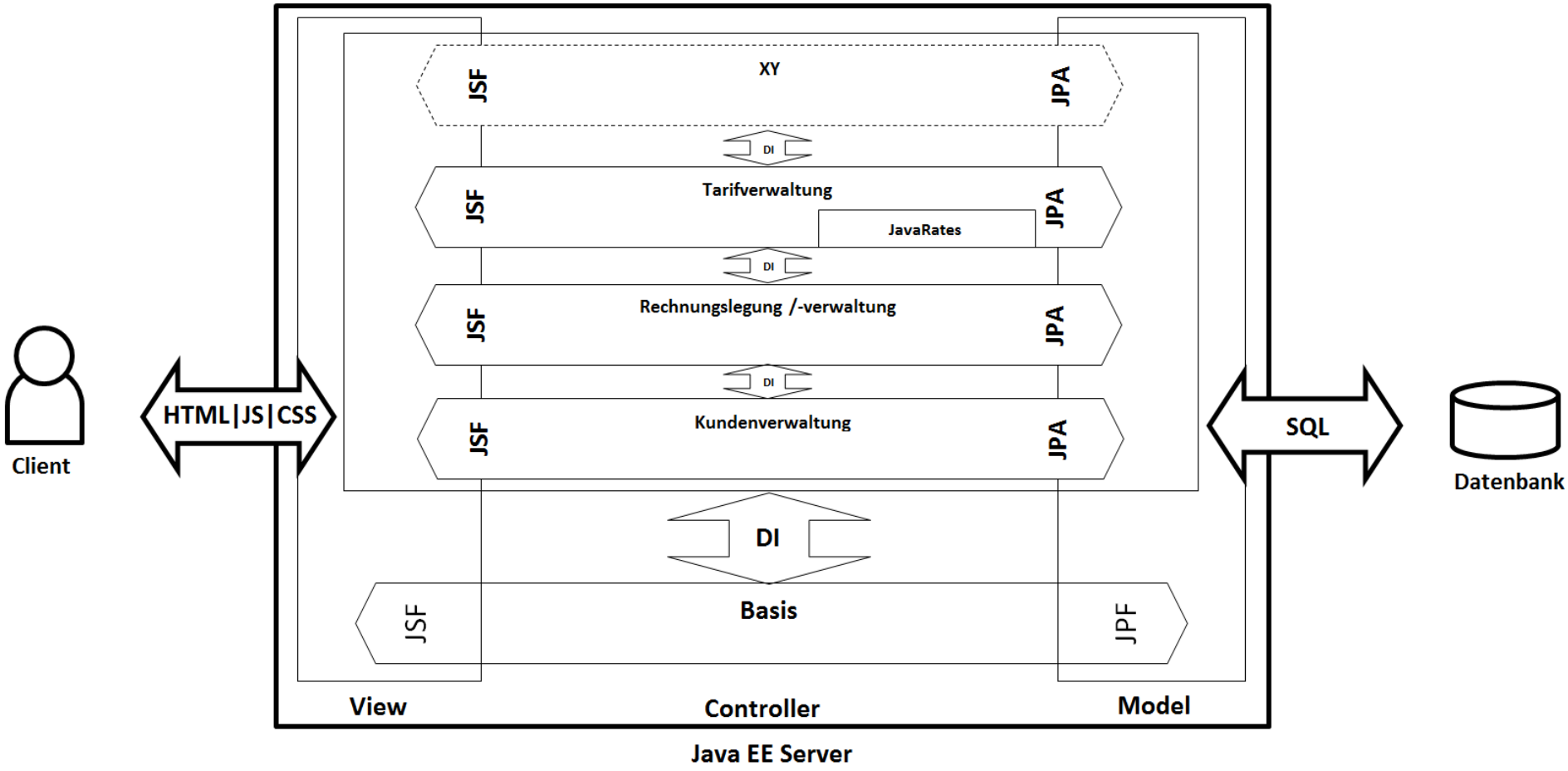


Abbildung 19 Gesamte Framework Architektur

4. Zusammenfassung

Im Zuge dieser Arbeit wurde zunächst festgestellt, dass Leistungen bzw. Produkte anhand von Tarifen im Zuge eines Verrechnungsprozesses verrechnet werden können. Durch die weitere Erkenntnis, dass Verrechnungsprozesse sehr variabel sein können und dass Softwaresysteme zur Abbildung solcher Verrechnungsprozesse ein durchaus großes Marktpotential haben, wurden in Folge weitere Fragen aufgeworfen. Die wichtigste aller Fragen war zunächst, was nun eigentlich ein Tarif ist. Hintergrund dieser Frage war auch die Zielsetzung, dass womöglich durch eine genauere Bestimmung der Tarife an sich, vielleicht auch ein abstraktes Modell bestimmbar wäre, welches man als Grundlage vieler solcher Verrechnungsprozesse verwenden könnte.

Daher war der Focus anfänglich auf die mathematische Beschreibung gerichtet, um in weiterer Folge eine hochabstrakte Implementation eines Tarifes in einer Hochsprache erstellen zu könne. Im Zuge der mathematischen Analyse wurde ein formales Modell entworfen, welches dabei half Grenzen einer praktischen Umsetzung zu definieren. Hierbei drängte sich auch die Frage auf, was eigentlich ein optimaler Tarif ist und wie er zu bestimmen sei. Durch eine anfängliche Einordnung in das große Feld der Optimierungsprobleme und weiteren Verfeinerungen, wie zum Beispiel die Erkenntnis, dass es sich hierbei um ein Vektoroptimierungsproblem handeln muss, wurde ersichtlich, dass die Suche nach optimalen Tarifen durchaus mit Problemen verbunden sein kann. Dennoch wurden einfache Wege aufgezeigt, wie man die Suche nach dem geschilderten Optimum gestalten könnte.

Alle gefundenen Erkenntnisse wurden im Anschluss dafür verwendet, eine Java Bibliothek zu entwickeln, die es ermöglicht Tarife abzubilden und zu berechnen. Diese Bibliothek erhielt den Namen JavaRates und wurde dem mathematischen Tarifmodell nachempfunden. Getroffene Designentscheidungen wurden anhand von Problemstellungen fundiert und mit Hilfe von Patterns entwickelt. Schlussendlich wurde die JavaRates Bibliothek auch als Grundstein für ein Modul des im letzten Kapitel dokumentierten konzeptionellen Frameworks verwendet.

Wie bereits oben angesprochen wurde, bestand der letzte Teil dieser Arbeit in der Beschreibung des konzeptionellen Frameworks. Auch hier wurden Desig-

nentscheidungen sowohl grundsätzlicher, als auch spezieller Natur durch Problemstellungen aufgezeigt und mit Hilfe von gängigen Konzepten und Patterns skizziert. Abschließend konnte durch Zuhilfenahme von Technologien rund um die Programmiersprache Java ein Framework entwickelt werden, das durch das Einbinden der JavaRates Bibliothek und der modularen Architektur durchaus als Grundlage einer breiten Masse an Verrechnungsprozessen dienen kann.

5. Literaturverzeichnis

[1] STATISTIK AUSTRIA, „LEISTUNGS- UND STRUKTURSTATISTIK Produktion & Dienstleistungen“, Wien: STATISTIK AUSTRIA, 2015. -ISBN 978-3-902925-77-0. S.1-264

[2] Cochran, William G., Sampling Techniques, 3rd ed., New York: John Wiley & Sons, 1977, ISBN 0-471-16240-X. S.1-17

[3] Ehrgott, Matthias: Multicriteria Optimization. Berlin Heidelberg: Springer Science & Business Media, 2006. -ISBN 978-3-540-27659-3. S. 7-8

[4] Ehrgott, Matthias: Multicriteria Optimization. Berlin Heidelberg: Springer Science & Business Media, 2006. -ISBN 978-3-540-27659-3. S. 65-124

[5] Hwang, Ching-Lai ; Yoon, Kwangsun: Multiple Attribute Decision Making : Methods and Applications A State-of-the-Art Survey. Wiesbaden: Springer Berlin Heidelberg, 1981. -ISBN 978-3-540-10558-9. S. 8

[6] Reinhardt, Rüdiger ; Hoffmann, Armin ; Gerlach, Tobias: Nichtlineare Optimierung : Theorie, Numerik und Experimente. 1. Aufl.. Berlin Heidelberg New York: Springer-Verlag, 2012. -ISBN 978-3-827-42948-3. S. 1-352

[7] Luenberger, David G. ; Ye, Yinyu: Linear and Nonlinear Programming. 4. Aufl.. Berlin, Heidelberg: Springer, 2015. -ISBN 978-0-387-74502-2. S. 333-475

[8] Goldberg, David: What every computer scientist should know about floating-point arithmetic. In: ACM Computing Surveys vols. 23 (1991), Nr. 1, S. 5–48

[9] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John ; Booch, Grady: Design Patterns : Elements of Reusable Object-Oriented Software. 1. Aufl.. Amsterdam: Pearson Education, 1994. -ISBN 978-0-321-70069-8. S. 1-457

[10] Buschmann, Frank ; Meunier, Regine ; Rohnert, Hans ; Sommerlad, Peter ; Stal, Michael: Pattern-Oriented Software Architecture, A System of Patterns. 1. Aufl.. New York: Wiley, 1996. -ISBN 978-0-471-95869-7. S. 1-457

[11] Java Platform Standard Edition 7 Documentation. URL http://docs.oracle.com/javase/7/docs/. - retrieved 2017-07-22. — Java Platform Standard Edition 7 Documentation

[12] Java Platform, Enterprise Edition: The Java EE Tutorial Release 7. URL https://docs.oracle.com/javaee/7/tutorial/. - retrieved 2017-07-22. — Java Platform, Enterprise Edition: The Java EE Tutorial Release 7

[13] Java Platform Standard Edition 8 Documentation. URL http://docs.oracle.com/javase/8/docs/. - retrieved 2017-07-22. — Java Platform Standard Edition 7 Documentation

[14] Tanenbaum, Andrew S. ; Steen, Maarten van.: Distributed systems principles and paradigms. Upper Saddle River : Prentice-Hall, 2007

[15] European Police Office (Europol): Internet Organised Crime Threat Assessment (IOCTA) 2016. Den Haag, Niederlande: European Police Office (Europol), 2016. –ISBN 978-92-95200-75-3. S.35-38

[16] Winkler, Reinhard: Logischer und mengentheoretischer Formalismus - Ärgernis, und sonst nichts? In: Didaktikhefte der ÖMG Nr. 42 (2010), S. 102–117

6. Abbildungsverzeichnis

Abbildung 1. Tarifmodell 21
Abbildung 2 Darstellung der Nebenbedingungen 41
Abbildung 3 Suche nach dem optimalen Extrempunkt 41
Abbildung 4 Gewichtungsvektor mit keiner eindeutigen Lösung 42
Abbildung 5. UML-Klassendiagram / Tarifeingaben, -ausgaben und –konfigurationen 54
Abbildung 6. Vektorwerte als erweitertes Objekt 55
Abbildung 7. Pseudocode nach Transformation 57
Abbildung 8 JavaRates Architektur 62
Abbildung 9 Verwendung eines JavaRates Tarifes unter Java 65
Abbildung 10 Stücktarif abgebildet als RateImplementation Klasse 66
Abbildung 11 UML-Sequenzdiagramm – JavaRates Aufruf 67
Abbildung 12 Tarifkomposition als gerichteter Graph 68
Abbildung 13 Tarifkomposition mit Hilfe von JavaRates 69
Abbildung 14 solve-Methode eines fiktiven Tarifes 72
Abbildung 15 optimize-Methode eines fiktiven Tarife 72
Abbildung 16 Java EE Architektur 83
Abbildung 17 Java EE Architektur als Layermodel 84
Abbildung 18 RateVector Objekt repräsentiert als XML String 87
Abbildung 19 Gesamte Framework Architektur 92

7. Anhang

Anhang 1

Louis Columbus Contributor

I cover CRM, Cloud Computing, ERP and Enterprise Software

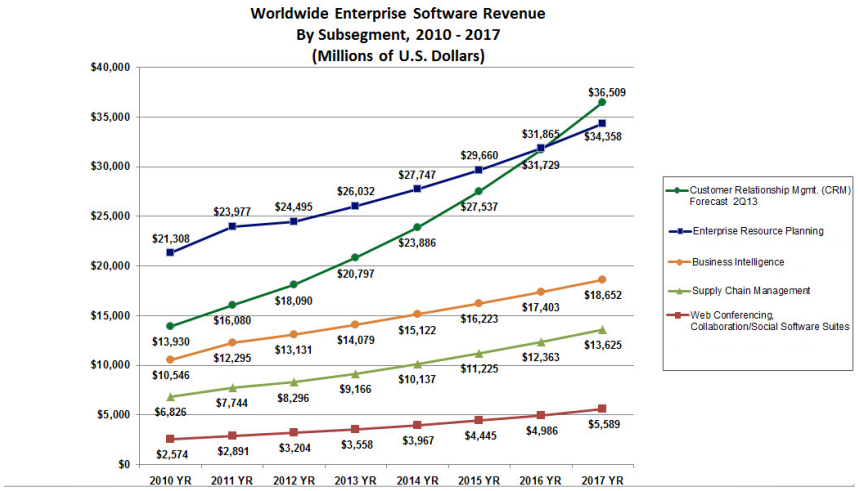
Opinions expressed by Forbes Contributors are their own.

TECH 6/18/2013 @ 12:25PM | 53.863 views

Gartner Predicts CRM Will Be A $36B Market By 2017

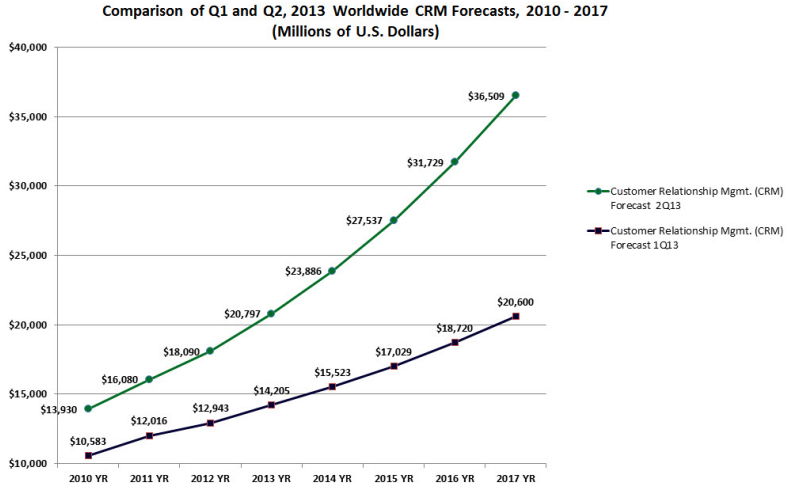
The latest enterprise software forecast from Gartner shows Customer Relationship Management (CRM) increasing to a $36.5B worldwide market by 2017, a significant increase from the $20.6B forecasted in Q1 of this year. CRM also leads all enterprise software categories in projected growth, showing a 15.1% CAGR from 2012 to 2017, also revised up from 9.7% in the Q1 forecast.

The latest round of forecasts published in the report, Gartner Forecast: Enterprise Software Markets, Worldwide, 2012-2017, 2Q13 Update shows CRM eclipsing ERP in worldwide market size in 2017. The following graph compares the relative growth of CRM, ERP, Business Intelligence (BI), Supply Chain Management and Web Conferencing, Collaboration/Social Software Suites. Source: Gartner Forecast: Enterprise Software Markets, Worldwide, 2012-2017, 2Q13 Update. Please click on the image to increase its size for easier reading.



Key Take-Aways

- Comparing Gartner's Q1 and Q2 CRM forecasts shows just how fast CRM growth is accelerating, netting a 56% increase in CAGR in the forecast period (2012 – 2017) from 9.7% in the Q1 forecast to 15.1% in the latest Q2 forecast. Sources: Gartner Forecast: Enterprise Software Markets, Worldwide, 2012-2017, 2Q13 Update and Enterprise Software Markets, Worldwide, 2012-2017, 1Q13 Update. Please click on the image to increase its size for easier reading.



- Worldwide enterprise software spending is projected to be $304B in 2013 in the latest forecast, up from $279B in the Q1 forecast. Gartner claims stronger demand for CRM, supply chain management and security are leading to accelerating market growth.
- ERP spending worldwide is projected to grow from $26.03B in 2013 to $34.3B in 2017, attaining a CAGR in the forecast period 2012 – 2017 of 7%.
- Business Intelligence (BI) worldwide is projected to grow from $14B in 2013 to $18.6B in 2017, attaining a CAGR in the forecast period 2012 – 2017 of 7.3%.
- Supply Chain Management (SCM) worldwide is projected to grow from $9.16B in 2013 to $13.6B in 2017, attaining a CAGR in the forecast period 2012 – 2017 of 10.4%.
- Data Integration Tools and Data Quality Tools worldwide are projected to grow from $4B in 2013 to $6B in 2017, attaining a CAGR in the forecast period 2012 – 2017 of 10.3%.

Bottom Line: Gartner's latest forecasts show that enterprises are realizing the most valuable assets they have are solid, long-term customer relationships. Trust really is the new currency, as my friend Michael Krigsman often says.

Anhang 2

vom 16.03.2015, 14:25 Uhr Update: 16.03.2015, 14:27 Uhr

Gewinner

Registrierkassen-Absatz könnte um bis zu 10 Prozent zulegen

Von WZOnline/APA

Olivetti-Geschäftsführer: Großteil der Kassensysteme kann ohne viel Aufwand nachgerüstet werden, Übergangsfrist wahrscheinlich.

Wien. Betrugsbekämpfung soll nach Regierungsplänen insgesamt 1,9 Mrd. Euro in die Staatskasse spülen, 900 Mio. Euro davon soll die "Registrierkassenpflicht samt technischer Sicherheitslösung" einbringen. Die Manipulation von Umsatzzahlen ist derzeit nämlich möglich und manipulationssichere Kassen ohne gesetzliche Vorschrift Ladenhüter, stellte noch die Steuerreformkommission in ihrem Bericht fest.

Die Registrierkassenpflicht soll nach derzeitigen Plänen der Regierung ab einem Nettojahresumsatz von 15.000 Euro schlagend werden und Branchen treffen, wo überwiegend bar bezahlt wird, insbesondere also Handel und Gastronomie.

Ausnahmen soll es für "kleine Vereinsfeste", "mobile Umsätze" (zum Beispiel Masseure) und "Gruppen mit kalten Händen" (etwa Maronibrater, Fiakerfahrer, Obst- und Gemüsehändler auf Bauernmärkten), also Händler ohne feste Verkaufsstellen, geben. Zusätzlich sind strengere Einzelaufzeichnungspflichten geplant, wie auch das obligatorische Ausstellen von Zahlungsbelegen.

Weil heute gängige Registrierkassen laut Steuerfahndern über Voreinstellungen verfügen, die eingebuchte Beträge nicht im Tagesumsatz speichern, wie auch über die Möglichkeit, Umsätze schon vor der Buchung zu frisieren, sollen Kassensysteme verpflichtend durch eine digitale Signatur manipulationssicher gemacht werden.

Als Vorbild dient hier das von der deutschen Physikalisch-Technischen Bundesanstalt entwickelte INSIKA-System ("Integrierte Sicherheitslösung für messwertverarbeitende Kassensysteme"). Angebot, Besitz und Verwendung von Manipulationsprogrammen sollen künftig strafbar werden. Für die Anschaffung eines neuen Kassensystems soll es eine 200-Euro-Prämie geben, eine sofortige Abschreibung der Investition soll nach Regierungsplänen möglich sein.

Walter Masten-Weber, Wiener Geschäftsführer des Registrierkassenherstellers Olivetti, erwartet, dass die Registrierkassenpflicht zu einer Absatzsteigerung von 5 bis 10 Prozent führen wird. Für die Branche sei die Registrierkassenpflicht ein Wachstumsschub.

Es sei jedoch abzuwarten, "ob nicht doch irgendwelche Gruppen ausgenommen werden". Da der Großteil der am Markt befindlichen Kassen ohne viel Aufwand mit einer Smartcard (Chipkarte) nachgerüstet werden könne, würden neue Kassen nur von jenen angeschafft, die jetzt noch keine haben. Einstiegsmodelle von "Fiskalkassen" seien ab 400 Euro zu haben, sagte Masten-Weber zur APA, die Hardwarekosten der Nachrüstung beliefen sich auf 30 bis 50 Euro.

Die Vorlaufzeit für die Umsetzung der Bestimmungen ist laut Masten-Weber grundsätzlich ausreichend, Olivetti habe bereits ähnliche Regelungen in Italien und Schweden implementiert. Falls die konkreten Durchführungsbestimmungen aber erst Mitte des Jahres kommen, werde es "verdammt knapp". Weil das auch bei früheren Gesetzesänderungen so gewesen sei, geht Masten-Weber davon aus, dass es auch diesmal eine Übergangsfrist geben wird. Olivetti beziffert seinen Marktanteil im Segment Handelskassen auf 25 bis 30 Prozent.

"Mit der Einführung der Registrierkassenpflicht schützen wir die ehrlichen Unternehmerinnen und Unternehmer gegen unlauteren Wettbewerb der Steuerbetrüger und liefern einen gerechten Beitrag zur Gegenfinanzierung der Steuerentlastung", kommentierte Staatssekretärin Sonja Steßl (SPÖ) den Regierungsentwurf am Samstag.

Wirtschaftskammer-Handelsobfrau Bettina Lorentschitsch fürchtet hingegen einen "Bürokratie-Overkill" und sieht vor allem kleine Betriebe aller betroffenen Branchen durch die Registrierkassenpläne schwer getroffen. Lorentschitsch zweifelt auch an der Höhe der erhofften Mehreinnahmen. Diese seien "vielleicht nicht um das Zehnfache, aber um ein Vielfaches überzogen".

Auf die Unternehmer kämen zusätzliche Kosten zu, kritisierte auch FPÖ-Tourismussprecher Roman Haider. "Ihnen wird damit auch unterstellt, bisher nicht ordnungsgemäß abgerechnet zu haben. Diese ständige Unterstellung der Unredlichkeit ist eine Frechheit."

Anhang 3

Die Renaissance der Registrierkasse

Foto: /Technisches Museum Wien Kassenschlager: Von einem misstrauischen Amerikaner vor langer Zeit erfunden

Sie wurde 1882 erfunden - und ist aufgrund der Steuerreform 2015 erneut sehr gefragt.

Uwe Mauch

27.03.2015, 05:00

Der Wiener Unternehmer Markus Zoglauer hat gut lachen: Er startet 2015 in eine neue Galaxie, in der auch ordentlich die Kasse klingelt. Seine Firma erzeugt nämlich hochwertige Registrierkassen-Systeme.

Klingelt es auch bei Ihnen? Ein Pfeiler der Steuerreform, die von Kanzler und Vizekanzler jüngst vorgestellt wurde, ist die Registrierkassen-Pflicht. Kommt sie, müssen laut Wirtschaftskammer 150.000 österreichische Betriebe so eine Rechenmaschine anschaffen. Kanzler und Vizekanzler sagen: Damit soll die Hinterziehung der Mehrwertsteuer eingedämmt, alles in allem 900 Millionen Euro eingespielt und damit auch zur Gegenfinanzierung der Reform beigetragen werden.

Die Betriebe und ihre Interessensvertreter maulen, lautstark. Man spricht von Generalverdächtigung und unangebrachtem Misstrauen. Dabei war es einst einer aus ihren Reihen, der die Registrierkasse erfunden hat. Aus Misstrauen.

Mitarbeiter-Kontrolle

Als Vater der Registrierkasse wird der Whisky- und Weinhändler James Jacob Ritty gehandelt. Der hat im Jahr 1871 in Dayton, Ohio, einen Saloon eröffnet: das Pony House. Der Schuppen lief gut, aber am Ende des

Tages soll dem Barbesitzer Bargeld gefehlt haben. Weil, wie Ritty meinte, seine Leute in die eigene Tasche arbeiteten.

Auf einer Schiffspassage nach Europa hatte Ritty die Idee seines Lebens: Beim Besichtigen der Dampfmaschine fiel sein Blick auf den Zeiger, der die Umdrehungen der Schiffsschraube anzeigte. Warum nicht für sein Pony House ein ähnliches Gerät bauen? Statt der Umdrehungen der Schiffsschraube könnte sein Automat die verkauften Getränke und Speisen anzeigen – und damit nicht zuletzt sein Personal kontrollieren.

Wieder in Dayton, baute Ritty eine Holzkiste mit Ziffernblatt und zwei Zeigern – für Dollar und Cent. Außerdem hatte seine Kiste zwei Reihen mit Tasten, die die Preisskala der Bar abbildeten. Nach jeder neuen Bestellung musste ein Hebel zum Addieren der Beträge betätigt werden. Beim Öffnen der Bargeld-Schublade klingelte es laut, wodurch den Gästen und vor allem dem misstrauischen Chef signalisiert wurde, dass gerade eine neue Bestellung registriert wird.

Nach der Sperrstunde zeigte die patente Kasse den Tagesumsatz an. Dank ihr ging es der Legende nach mit dem Pony House steil bergauf. Und die Registrierkasse wurde bald zu einem US-Exportschlager.

Hochzeit mit dem PC

Foto: /Juergen Skarwan

Hundert Jahre später ging der Steirer Ewald Strametz daran, die Registrierkasse mit dem Computer zu verheiraten. Strametz hatte in den 1980er-Jahren in seinem Fotostudio in Deutschlandsberg viel Kundenverkehr. Der PC war gerade leistbar geworden. Und seine technische Lösung, die er für den Eigenbedarf entwickelt hatte, wurde bald auch von Kollegen angefragt.

Zurück in der Gegenwart. Markus Zoglauer hat vom steirischen Pionier die Lizenzen gekauft. Seit nunmehr zwanzig Jahren baut er mit seiner Firma Etron das Geschäft kontinuierlich aus: "Wir hatten zu Beginn gerade einmal zwanzig Kunden, heute beliefern wir insgesamt 2000 Standorte."

Zu Zoglauers Stammklientel zählen heute vor allem Fachhändler: weiterhin die Fotografen, aber auch Spielzeug-, Papier-, Elektro-, Leder-, Schuh-, Sportartikel- und Fahrrad-Verkäufer sowie die Vielzahl an Tabak-Trafikanten.

Wo die Kasse klingelt

Begonnen hat der Etron-Gründer mit einem Kompagnon, heute hat er dank fleißiger Akquise 20 Angestellte, darunter vier Programmierer, sowie eine kleine Armada von Vertriebsleuten. Und das ist noch nicht einmal das Ende der Geschichte.

Anders als die Funktionäre der Wirtschaftskammer sieht der Kassen-Mann die Steuerreform als Chance: Für Tierärzte, Ärzte, Physiotherapeuten, Ab-Hof-Verkäufer, Marktfahrer, die mehr als 15.000 Euro pro Jahr netto umsetzen. Sie alle könnten mit einer Registrierkasse ihre Kunden und auch ihr Lager besser überblicken.

Er selbst will zusätzlich Personal aufnehmen. Denn es wird wohl auch seine eigene Kasse klingeln. Die Registrierkassenpflicht in Österreich (übrigens in den meisten Nachbarländern bereits Standard) sei für ihn "eine Art Probegalopp": für eine geplante und in eine ähnliche Richtung zielende Gesetzesänderung in Deutschland.

(kurier) Erstellt am 27.03.2015, 05:00

Anhang 4

Registrierkassenmesse: "Es trifft immer die Kleinen"

VIDEO
TEXT: SIMON MOSER | VIDEO: MARIA VON USSLAR, SIMON MOSER & ALEXANDER GOTTER
2. März 2016, 15:27

1278 POSTINGS

Viele Kleinunternehmer zeigen sich frustriert. Je älter die Firmenchefs, desto größer der Unmut

Die Wiener Sofiensäle sind ein eigentümlicher Ort: Das ehrwürdige Hauptportal mitsamt dem renovierten großen Saal wird beinahe erdrückt vom modernen Neubau, der seit einigen Jahren an der Stelle des 2001 bis auf die Grundmauern abgebrannten historischen Bauwerks steht.

Den Eindruck, das Etablierte drohe mancherorts vom Modernen überwältigt zu werden, teilen auch viele Besucher, die diese Woche zur Registrierkassenmesse der Wirtschaftskammer in die Sofiensäle pilgern. Der Andrang ist groß: die rund 60 Aussteller männlich dominiert, das Publikum gemischt, aber tendenziell im höheren Alter. Und je älter die Unternehmer, wird im Gespräch schnell klar, desto mehr Probleme scheinen sie mit der Registrierkasse zu haben.

usslar
Die Kasse als Symbol für den angestauten Unternehmerfrust.

Der Besitzer eines Eissalons, italienische Wurzeln, um die 60, klagt über fehlende IT-Kenntnisse, die ihm die Wahl der passenden Kassenlösung vergällen. Für die ältere Generation sei das ein Problem, ebenso die Registrierkassenpflicht im Allgemeinen. "Mehr Arbeit, die niemandem etwas bringt außer dem Finanzamt, wenn überhaupt." Er ist froh, dass er einen Anbieter mit gutem Service gefunden hat – 2.600 Euro kostet das Kassensystem. "Das musst du als kleine Firma auch erst einmal verdienen."

Ein anderer älterer Herr – er ein Ein-Mann-Detektivbüro, die Tochter Gastronomin – hält ebenso wenig von den neuen Regeln gegen die Umsatzsteuerverkürzung: "Für Kleinunternehmer ist das ein Riesenschaden. Das kostet nur Energie und Geld."

Geballte Ablehnung

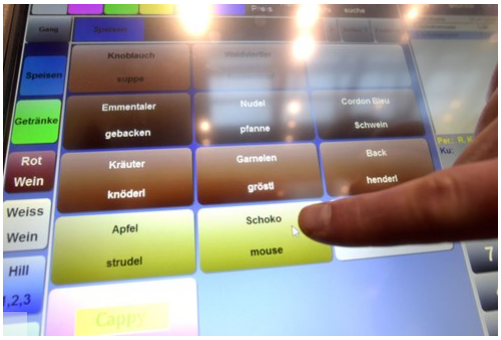
Kleinunternehmer wie diese sind es, die neben der Alterskorrelation einen zweiten Zusammenhang zutage treten lassen: Der Unmut der Unternehmer, er scheint sich umgekehrt proportional zur Größe des Betriebs zu verhalten. "Es trifft immer die Kleinen", hört man hier öfter, ebenso das Wort "Schikane".

Überfordert fühlt sich auch eine Nagelstudiobesitzerin, die sich als Smartphone-Verweigerin outet und in ihrem Geschäft über keinen PC verfügt. "Furchtbar, eine Katastrophe. Ich will einfach nur meine Arbeit machen und mich nicht mit diesem Blödsinn beschäftigen müssen", sagt die Wienerin, die mit rund 500 Euro Investitionskosten rechnet.

Atmosphärische Un-Messe

An der Registrierkasse, dem Symbol für eine vermeintliche Gängelung durch den Staat, entlädt sich ein länger aufgestauter Unternehmerfrust. Zumindest gefühlt ist die Belastung höher als die Summe der verwaltungsrechtlichen Auflagen.

Fragt man nach, was genau das Schlimme an der Kasse ist, sind sich die Betroffenen uneins. Die Bedienung sei einfach, aber die Kosten hoch, meint einer. Andere sehen Komplettlösungen um wenige hundert Euro als finanziellen Klacks. Aber die Bedienung und der Aufwand rund um die Einführung! Gerne scheint jedenfalls kaum jemand hier zu sein, man wünscht sich atmosphärisch auf eine Reisemesse.



matthias cremer
Schokomousse oder Apfelstrudel? Für alles gibt es eine Taste.

Ein schlechtgelaunter Mietwagenunternehmer – vier Wagen umfasst seine Flotte – löchert den Anbieter von speziellen Taxikassen. "Für 700 Euro bekomme ich nur die Software, das ist alles?" Vom Offert ist er wenig angetan. Noch einmal 150 Euro kostet der Bondrucker, den er dazukaufen müsste. Für des Mietwagenbesitzers Smartphone hat der Softwareanbieter noch keine Anwendung fertigprogrammiert – "der Chef arbeitet noch dran". Der Kunde bleibt ein potenzieller.

Großer Beratungsbedarf

Wie er nehmen die meisten hier abgesehen von haufenweise Broschüren nichts Handfestes mit nach Hause. "Die Leute kommen, um sich zu informieren, nicht, um zu kaufen", sagt Harald Lindenbauer, Geschäftsführer von XMT Kassensysteme aus Oberösterreich. "Wenn ich mich selbst in ihre Lage versetze – ich wäre auch überfordert." Der Kunde nehme den Prospekt mit, vergleiche in Ruhe, rufe dann bei näherem Interesse an und wünsche ein individuelles Beratungsgespräch.

Auslieferung, Installierung und Einschulung für die vielen Spätentschlossenen unter den Kunden können für Anbieter zu einer echten Herausforderung

werden. Denn die Zeit drängt: Zwar gilt für die Finanz bis Juli noch der Grundsatz "Beraten statt strafen". Jedoch muss man im Fall von Kontrollen schon ab April vorweisen können, dass man sich zumindest darum bemüht hat, eine Kasse anzuschaffen. Wer etwa glaubhaft machen kann, dass nur aufgrund von Lieferengpässen noch keine Kasse vor Ort ist, entgeht einer Verwaltungsstrafe.

matthias cremer

Rund 60 Aussteller präsentieren sich auf der Messe.

Drohende Lieferschwierigkeiten

Es sei sowohl für die Kunden als auch für die Anbieter ein "Riesenproblem", dass sich die Kunden so spät entscheiden, sagt Lindenbauer. Auch andere Aussteller klagen darüber, dass sie nur eine begrenzte Anzahl an Geräten auf Lager haben – seien es PCs, Kassenschubladen, Barcodescanner oder Bondrucker. Die meisten bestellen bei einer Handvoll Großhändler, die die Hardware ausliefern. Engpässe seien also programmiert, sagen einige.

Andere dagegen sehen das mit dem Zeitdruck entspannter. Die drei zusätzlichen Monate bis Juli, in denen bei Vorlage einer Bestellbestätigung nicht gestraft wird, seien genug Zeit, um alle Bestellungen abzuarbeiten, sagt ein Aussteller.

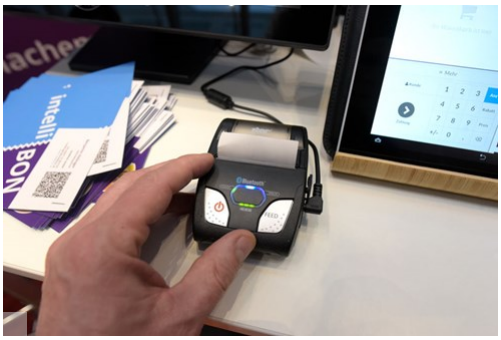
Boomender Markt

600 Kassensystemanbieter listet die Wirtschaftskammer derzeit auf ihrer Website. Weniger als 100 waren es laut Markus Knasmüller, Sachverständiger für Kassensoftware, noch vor einem Jahr. Ein Boom, den selbst Experten nicht für möglich gehalten haben. Unseriöse Anbieter, die beispielsweise keine Aktualisierungen im Rahmen der noch zu erwartenden Änderungen an den technischen Auflagen für die Softwarelösungen garantieren, gebe es nur vereinzelt, sagt Knasmüller. "Hände weg von solchen Anbietern", rät er. Auf der Messe stellen ausschließlich Anbieter aus, die diese Sicherheit bieten.

Unternehmensberaterin Regina Wengenroth ist weder als Kundin noch als Herstellerin auf der Veranstaltung, sondern in eigener Mission. Es brauche unabhängige Berater, um die Lawine an Informationen zu sondieren, die über die Unternehmer hereingebrochen sei. Was das richtige Angebot für das je eigene Geschäftsmodell ist, sei für den Einzelnen nur unter großem Aufwand herauszufinden. Kleinunternehmer aus allen Branchen hätten die größten Probleme, so Wengenroth.

Viele Last-Minute-Käufer

Was unter vorgehaltener Hand von den Anbietern mehrfach zu hören ist: dass Unternehmern sowohl von Steuerberatern als auch von der Wirtschaftskammer lange gesagt wurde, sie sollten sich mit dem Einholen von Informationen noch Zeit lassen. Eigentlich sei die Regelung, die ja schon im Sommer beschlossen wurde, zeitlich nicht zu knapp bemessen gewesen. Unternehmensberaterin Wengenroth, die zur Kassenverordnung Info-Veranstaltungen abhält: "Als Unternehmer, was würden Sie machen, wenn Ihnen jemand sagt: Warten Sie noch mit der Investition?"



matthias cremer

Ein Minidrucker für die Rechnung beim Wirt.

Die Wirtschaftskammer hingegen argumentiert damit, dass das Finanzministerium die formellen Vorgaben für die Kassensoftware erst knapp vor Jahreswechsel ausgearbeitet hat. Anbieter hätten schlecht etwas anbieten können, was noch gar nicht spezifiziert ist, so die Kammer.

Umfangreiche Zusatzfunktionen

Jenen Firmenchefs, die noch nie mit digitalen Kassen gearbeitet und sich bisher auf den Rechnungsblock verlassen haben, dürften diese Details großteils egal sein. Mehrfach hört man auf der Messe, dass jene Unternehmen, die sich über Zusatzfunktionen wie die Dateneinbindung in ein Warenwirtschaftssystem, Schnittstellen zu einem bereits vorhandenen Buchhaltungssystem oder eine umfassenden Stammkundenverwaltung Gedanken machen, tendenziell früher dran waren. Last-Minute-Kunden sind eher jene, die einfache Geräte suchen.

Sie müssen sich für eine von drei grundsätzlichen Kassenarten entscheiden: die klassische Hardware mit Kassenschublade, eine Softwarelösung für den PC oder eine App für Smartphone und Tablet. Bei letzterem System wird eine dauerhafte Internetverbindung benötigt. Fällt die Verbindung einmal aus, kann eine handschriftliche Rechnung ausgestellt und die Transaktion nachträglich in die Kasse eingetippt werden. Knasmüller empfiehlt beim Vergleich eine Durchrechnung über fünf Jahre, um die kostengünstigste Lösung zu finden.

"Unzumutbar"

Thomas Anderl ist Eigentümer eines Marktstandbetriebs, er verkauft auf dem Wiener Yppenmarkt und an fünf anderen Standorten einmal pro Woche regionale Bioprodukte. Jahresumsatz: rund eine halbe Million Euro. Weil auf dem Markt eine hohe Kundenfrequenz herrsche, breche das Arbeitstempo ein, greife man nicht zu einer hochwertigen, also teuren Lösung. Für alle seine Standln braucht er eine eigene Kasse mitsamt Waage. 3.000 Euro pro Exemplar plus 900 Euro für die Software, so hoch seien die Anschaffungskosten. Für alle Marktstände zusammen mache das mindestens 20.000 Euro aus, rechnet er vor. "Unzumutbar", sagt Anderl.

Günstigere Tablet-Lösungen hat er sich auf der Messe auch schon angesehen. Die seien aber weder temperatur- noch wetterfest, auch könne man damit weniger schnell arbeiten. "Wir stehen draußen teils bei minus zehn Grad. Da steht jeder Thermodrucker", spricht er technische Einschränkungen an. "Und sehr viel billiger wird's dann auch nicht." Die Registrierkassenpflicht im Allgemeinen? "Eine große Frechheit."

Unterschiedliche Wahrnehmungen

Wie zur Bestätigung, dass sich jüngere Semester leichter tun, findet sich gleich daneben ein optimistischer Mittdreißiger, dem die Registrierkassenpflicht keine schlaflosen Nächte beschert: "Ich plane, ein Lokal zu eröffnen, und es schaut so aus, also ob alle Systeme sehr ähnlich sind, auch vom Preis her." Ob App- oder Hardwarelösung, sei noch nicht entschieden, Hindernis für die Gründung sei die Kasse aber jedenfalls keines. Mit 2.000 Euro Anschaffungskosten rechnet der angehende Barbetreiber. Dass irgendjemand zusperren muss, weil er sich eine Registrierkasse anschaffen muss, kann er sich nicht vorstellen.

Nur ein paar Meter weiter will ein Aussteller eine skeptisch blickende ältere Dame animieren, zu seinem Stand zu kommen. Ihre aufrichtige Quittung: "Na, danke. Ma is ja sowieso überfordert mit allem." (Text: Simon Moser, Video: Maria von Usslar, Simon Moser & Alexander Gotter, 2.3.2016)

Die Registrierkassenmesse der Wirtschaftskammer Wien findet noch bis Samstag statt, jeweils von 13 bis 21 Uhr. Eine Anmeldung ist nicht erforderlich, der Eintritt ist frei.

Mehr zur Kassenpflicht:

Pannonisches Regierungscash für Wirtenkassa

Klage gegen die ungeliebte Kasse

Registrierkassenpflicht: Was auf Unternehmer zukommt

Anhang 5

STEUERREFORM

Registrierkassenpflicht: Gutes Geschäft für Hersteller

Erst wenn technische Details der Steuerreform geklärt sind, zeigt sich, welche Kassen umgerüstet werden müssen. Finanz bekommt hunderte neue Mitarbeiter für die Betrugsbekämpfung.

Von **Roman Vilgut** | 06.00 Uhr, 19. März 2015

Noch ist nicht klar, welche Kassen in der Gastronomie umgerüstet werden müssen © APA/ROLAND SCHLAGER

Registrierkassenpflicht?! Die Wirtschaft bringt diese neue Vorschrift zur Weißglut. Von Generalverdacht und Bespitzelung ist die Rede. Doch es gibt auch die Gewinner der Kassenpflicht. Heimo Högl zum Beispiel. Sein Familienunternehmen verkauft seit 30 Jahren Registrierkassen in Südösterreich. Ob die Ausweitung der Kassenpflicht dem Staat wirklich 900 Millionen Euro bringt und wie teuer die Aufrüstung für die Unternehmer wird, hänge von den noch offenen technischen Details ab, sagt Högl.

Bereits 2012 wurde eine Kassenrichtlinie erlassen. Sie schreibt vor, dass elektronische Anlagen die Buchungen speichern und maschinell lesbar machen – gültig für Firmen ab einem Jahresumsatz von 150.000 Euro. Doch nicht alle Geräte sind manipulationssicher. Eine Möglichkeit, das zu gewährleisten: Das Finanzamt schreibt eine bestimmte Chipkarte für diese Systeme vor. Högl: „Dann müssten aber auch ältere Registrieranlagen ausgetauscht werden, die keine Schnittstelle für eine Karte haben." Solche Systeme kosten zwischen 200 und 400 Euro.

Eine zweite Variante wäre eine direkte Verbindung mit dem Finanzamt. Doch dies sei nur mit hohem Aufwand zu bewerkstelligen, sagt Högl. „Jede Buchung würde eine verschlüsselte Verbindung brauchen und eine nachvollziehbare Belegnummer." Außerdem müsste die Kasse eine ständige Anbindung ans Internet haben - in ländlichen Regionen kann das ein Problem sein. Die Anschaffungskosten dieser Kassen wären mit 2000 bis 3000 Euro zu beziffern. Hinzu kommt, dass auch das Finanzministerium aufrüsten müsste. Högl: „Da würden eine Million Kassen im Sekundentakt auf die Server des Ministeriums zugreifen." Högl erklärt, dass Registrierkassen von seinen Kunden nicht als Belastungen wahrgenommen werden. Sie seien ein willkommenes Hilfsmittel. „Heute merkt sich kaum noch jemand Preise, und auch Kopfrechnen kann man oft nicht mehr verlangen. Für den Steuerberater wird es einfacher. Mit einem Knopfdruck kommt man zur Monatsabrechnung."

Walter Masten-Weber, Geschäftsführer des Kassenherstellers Olivetti Österreich, hofft auf eine Absatzsteigerung um bis zu zehn Prozent. Högl glaubt allerdings, dass Übergangsfristen nötig sind. „Erst wenn das Finanzministerium Vorgaben bekannt gibt, können die Produzenten die nötigen Hard- und Softwareanpassungen machen. Das dauert dann bis zu einem halben Jahr."

500 neue Finanzprüfer

Registrierkassen sind allerdings nur ein Teil der Maßnahmen gegen Steuerbetrüger. Auch das Personal wird aufgestockt. Laut Finanzministerium sollen 500 zusätzliche Prüfer kommen. Wo die neuen Mitarbeiter eingesetzt werden, wird noch geklärt. Die höheren Lohnkosten zahlen sich aus. Laut Rechnungshof spielen Steuerprüfer ihr Gehalt in mehrfacher Höhe wieder ein.

Deutlich höher als zunächst erwartet dürften die Einnahmen aus der neuen Grunderwerbssteuer sein. Josef Schmiedinger, Chef der s-Bausparkasse, geht von 300 Millionen Euro aus. Doch nicht nur die Steuer macht Erben und Schenken teurer. Für die Feststellung des Verkehrswerts müsse ein Sachverständiger beigezogen werden. Zusatzkosten: 800 bis 1200 Euro.

ROMAN VILGUT

MEHR ZUM THEMA

AUFLAGENDSCHUNGEL
Gastronomie: "Wenn der Amtsschimmel wiehert"
(/wirtschaft/4688394/Auflagendschungel_Gastronomie_Wenn-der-Amtsschimmel-wiehert)

Roman Vilgut
Wirtschaftsredakteur
(mailto:roman.vilgut@kleinezeitung.at)

Mehr von Roman Vilgut ›

Anhang 6

Abschlepp- und Bergedienst
Telefonnummer Büro 0664/42 39 455
24h Hotline 0664/388 61 83

w w w . a b s c h l e p p e n - b e r g e n . a t

Preisliste inkl. Ust

Abschleppdienst Tag

Mo- Fr 07.00 – 18.00 Uhr

Ausfahrtspauschale inkl. 20 km Gesamtstrecke	€ 80,-
Kilometerpreis	€ 1,50
Mautkosten	€ 0,20

Abschleppdienst Nacht

Mo- Fr 18.00 – 07.00 Uhr, Sa, So, FT 0-24 Uhr

Ausfahrtspauschale inkl. 20 km Gesamtstrecke	€ 100,-
Kilometerpreis	€ 2,-
Mautkosten	€ 0,20

Bergung

Bergefahrzeug mit Kran (Mindestverrechnung 1 Stunde)	€ 120,-
Bergungsleiter (Mindestverrechnung 1 Stunde)	€ 85,-
Bergehelfer (Mindestverrechnung 1 Stunde)	€ 42,-
Anschlagmittel (Gurte, Ketten, etc)/Stück	€ 6,-
Reinigung Bergefahrzeug	€ 42,-

Fahrzeugverwahrung

Halle	€ 15,-
Freigelände (nur möglich wenn kein Flüssigkeitsverlust)	€ 10,-

Pannenhilfe

Tagsüber inkl 20 km Gesamtstrecke	€ 120,-
Nacht/Wochenende inkl 20 km Gesamtstrecke	€ 180,-
Kilometerleistung Pannenhilfe	€ 1,-

Mietwagen

Preis pro Tag (24 h Verrechnung), inkl. 200 km	€ 60,-
Mehrkilometer	€ 0,25

Abschleppdienst Machalek Harald, Pernersdorf 159, A-2052 Pernersdorf
www.abschleppen-bergen.at office@abschleppen-bergen.at
Tel.: 0664/42 39 455 Fax: 02944 20 008
Bankverbindung: Raika Retz-Pulkautal BLZ: 32715 Kontonr.: 76.257 H. Machalek
UID NR.: ATU62361606