Technische Universität Graz
Institut für Maschinenbau- und Betriebsinformatik

Masterarbeit

Token Based Lizensierungsstrategien

eingereicht von

Andreas Eibler

Gutachter: Univ.-Prof. Dipl.-Ing. Dr.techn. Siegfried Vössner
Betreuer: Dipl.-Ing. Sabine Hösch, Ing. Dipl.-Ing. Gerhard Tieber

Graz, 2012

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 entnommene Stellen als solche kenntlich gemacht habe.

Graz, am ....

(Unterschrift)

Abstract

Software licensing steadily becomes a more important topic in industrial environments. Increased offerings for software solutions result in customers having a broader variety of software contractors to choose from. Software contractors reacted by increasing their offered software portfolio to raise attractiveness, diversification, customer bonding and overall competitiveness. The disadvantage of having a broader field of software solutions is, that customers are not able to make use of the portfolio as a whole. Therefore they are not willing to invest into rarely used parts of the portfolio. A more dynamic approach to licensing than the traditional time based model had to be developed. A solution that combines dynamic usage calculation but still offers the advantages of periodical accounting, is Token Based Licensing. This masters thesis adresses the problem of translating a traditional time based licensing model into a token based model. A method was developed to evaluate single software solutions in a software portfolio and create a dynamic token based licensing system. This method was developed in behalf of the firm AVL to expand or substitute their existing licensing model with the use of user data provided by the company. During the analysis several key faktors which influence a token based licensingmodel were found to benchmark and compare different softare solutions.

Key words: Token Based Licensing, Licensing model, Software licensing, on-demand licensing

Zusammenfassung

Softwarelizensierung im industriellen Bereich spielt eine immer stärker werdende Rolle. In bestimmten Domänen ist die Anzahl der Anbieter für Softwarelösungen in der Vergangenheit stetig gestiegen. Lizenznehmer können auf Grund dessen zwischen einer Vielzahl an Angeboten wählen. Um weiterhin konkurrenzfähig zu bleiben, griffen viele Anbieter auf Produkterweiterungen zurück. Diese Produkterweiterungen sollten ein umfassenderes Angebot schaffen, sich von der Konkurrenz abheben helfen und den Kunden stärker binden. Dadurch entstanden breit gefächerte Softwarepakete, die eine Vielzahl an artverwandten Funktionen abdecken. Dies führte aber im Gegenzug dazu, dass man zwar ein breites Spektrum an Kunden bedienen konnte, aber große Teile der Software wenig- oder ungenutzt blieben bzw. Kunden große Teile des Angebots aus Kostengründen nicht lizensieren wollten. Ausgehend von den starren, traditionellen Softwarelizensierungsmodellen mussten Alternativen entwickelt werden, die es dem Kunden ermöglichen, individuell und dynamisch das angebotene Softwareportfolio nach Bedarf zu nutzen. Eines dieser dynamischen Lizensierungsmodelle ist Token Based Licensing. Bei diesem Lizensierungsmodell wird angestrebt, dynamisch auf die Bedürfnisse der Kunden einzugehen, während gleichzeitig eine feste Abrechnung über eine Lizensierungsperiode erfolgt. In dieser Masterarbeit wird eine Methode erarbeitet, die zur Bewertung von einzelnen Softwarelösungen herangezogen werden kann und diese in Relation zueinander setzt. Anhand dieser Bewertung kann, in weiterer Folge, ein Token Based Licensing System aufgestellt werden. Diese Methode zur Erstellung eines Token Based Lizensierungsmodells wurde durch die Firma AVL in Auftrag gegeben, um das zur Zeit praktizierte Modell zu ergänzen oder zu ersetzen. Während der Analyse der Daten wurden mehrere Einflussfaktoren definiert, die eine wichtige Rolle bei der Erstellung eines Token Based Lizensierungssystems wichtig sind, und ebenso für den Vergleich von Token Based Modellen von Nutzen sind.

Stichworte: Token Based Licensing, Lizensierungsmodell, Softwarelizenz, dynamische Lizensierung

Inhaltsverzeichnis

Kapitel 1

Einleitung und Ausgangssituation

In diesem Kapitel werden die Beweggründe der Firma AVL erläutert, die Anstoß für diese Masterarbeit gegeben haben. Es wird auf die Motivation, warum ein neues Lizensierungssystem angestrebt wird, eingegangen und welche Vor- und Nachteile eine Änderung mit sich bringt. Es wird ebenfalls auf die Ausgangssituation eingegangen, in der sich die Firma AVL mit ihrem Softwareportfolio befindet. Abschließend wird erläutert, wie die Arbeitsschritte dieser Arbeit unterteilt und durchgeführt wurden.

1.1 Motivation dieser Arbeit

Ausgangspunkt für diese Masterarbeit, die von der Firma AVL in Auftrag gegeben wurde, war der Wunsch nach einem dynamischeren Lizensierungssystem für das angebotene Softwareportfolio. Ein Softwareportfolio besteht in der Regel aus einer Vielzahl an einzelnen Softwarelösungen, auch Features genannt, die selbständig funktionieren, aber dennoch in einer Beziehung zueinander stehen. Es wird angestrebt, eine möglichst große Anzahl an Kunden bedienen zu können. Aus diesem Grund wird der Umfang des Softwareportfolios ständig erweitert, um somit das Angebot zu verbessern. Aus diesem breit gefächerten Angebot entstehen jedoch folgende Probleme. Einzelne spezialisierte Unternehmen benötigen nur ausgewählte Teile des Softwareportfolios. Aus diesem Grund werden die Softwareprodukte in Pakete und weiter in einzelne Lösungen unterteilt und separat lizensiert. Nun kann individuell auf Bedürfnisse des Kunden eingegangen werden, da dieser ausschließlich die Software lizensiert, die benötigt wird. Diese Art der Lizensierung ist zwar an den Kunden angepasst, aber dennoch nicht dynamisch. Der Kunde ist nach wie vor gezwungen, eine fixe Anzahl an Lizenzen aus seinem Bereich für einen festen Zeitraum (üblicherweise

ein Jahr) zu erwerben. Diese Lizensierungsmethode schafft Unzufriedenheit bei Kunde und Anbieter. Ein Kunde hat keinerlei Möglichkeit kurzfristig auf geänderten Bedarf zu reagieren. Die Bedarfskalkulation muss immer jährlich erfolgen. Dies führt dazu, dass bei hohem Bedarf Mitarbeiter keinen Zugang zu Lizenzen haben und warten müssen, bzw. bei geringer Auslastung, die Lizenzen brach liegen. Anbieter haben das Problem, dass angebotene Software, die in Nischenbereichen Anwendung findet, nur selten lizensiert wird, obwohl viel Zeit und Geld in Entwicklung geflossen ist. Da der Bedarf über den Zeitraum eines Jahres gerechnet wird, sparen Kunden zuerst bei Software, die nur selten gebraucht wird. Eine dynamischere Lizensierungsmethode kann hier Abhilfe schaffen. Je dynamischer ein Lizenzmodell ist, desto mehr Freiheit lässt es dem Kunden bei der Nutzung der Software. Bei einer komplett freien Methode, dem Pay-Per-Use, kann ein Kunde, je nach Situation, unterschiedlich viele Lizenzen benutzen, ohne eingeschränkt zu sein. Mitarbeiter, die auf eine freie Lizenz warten müssen, oder Ressourcen, die brach liegen, gibt es bei dieser Methode nicht. Die Pay-per-use Methode hat aber den entscheidenden Nachteil, dass sie nur im Nachhinein abgerechnet werden kann. Aus diesem Grund ist diese Methode der Lizensierung nicht sehr häufig anzutreffen. Firmen ziehen Lizensierungsmethoden vor, bei denen die Kosten bzw. Einnahmen vorab bekannt sind. Einen Mittelweg soll Token Based Licensing eröffnen. Diese Lizensierungsmethode stellt einen Kompromiss aus jährlichen, fixen Lizenzen und dynamischer Softwarenutzung dar.

1.2 Ausgangssituation und Problemstellung

Die Firma AVL besitzt eine Sparte für Simulationstools (AVL - Advanced Simulation Technologies). Deren Angebot umfasst Softwareprodukte im Bereich der Automobilindustrie. Die AVL AST hat im Bereich physikalische Simulationen, Prototyping, Design und Tests Expertise aufgebaut. Ein gesamter Entwicklungsprozess für Automobile kann mit diesem Angebot abgedeckt werden. Angeboten werden vier Pakete, die jeweils weiter in Lösungen unterteilt sind. Jede Lösung ist in drei Features aufgeteilt: Preprocessor, Postprocessor und Solver. Die Lizensierungsmethode für die Features basiert auf fixen Jahreslizenzen. Aufgrund der hohen Anzahl der Lösungen innerhalb eines Pakets war es Kunden nur selten möglich das gesamte Produktportfolio auszuschöpfen. Die Kosten für die Lizensierung von selten genutzten Softwarelösungen würden den Nutzen übersteigen. Folglich wurde der Ruf nach einer dynamischeren Lizensierungsmethode laut. Es wurde angestrebt, anhand der durch die von der Firma AVL bereit gestellten Benutzerdaten ein Bewertungsschema

zu finden, um die einzelnen Softwareprodukte zu klassifizieren. Die Klassifikation sollte anhand der durch die Software gegebenen Eigenschaften erfolgen. Mittels der erhaltenen unterschiedlichen Softwareklassen, die aus diesem Verfahren resultieren, sollte ein Schema erstellt werden, welche diese klassifizierten Produkte miteinander kombiniert. Die dadurch kombinierten Produkte sollten eine Tokensetup (Kombination verschiedener bewerteter Softwareprodukte) ergeben, das dem Kunden eine dynamischere Lizensierungsvariante bietet, aber dennoch für die Firma AVL ein profitables Geschäftsmodell darstellt.

1.3 Vorgehensweise

Diese Arbeit kann in vier große Arbeitsschritte unterteilt werden. Diese Arbeitsschritte sind in Abbildung 1.1 ersichtlich:

- Schaffen einer fachlichen Basis durch Recherche zum Thema Lizensierung.
- Analyse der AVL Daten und Ermittlung eines Verfahrens zur Überführung in ein neues Lizensierungsmodell.
- Finden von Bewertungs- und Einflussgrößen, um ein Token Based Lizenzsmodell zu berechnen und zu vergleichen.
- Anwendung der Ergebnisse aus dem Experiment.

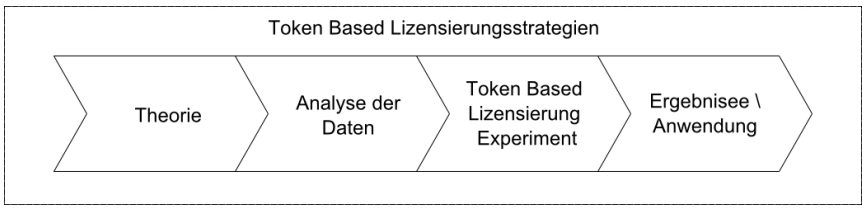

Abbildung 1.1: Darstellung der Unterteilung der Arbeit in vier Teile

1.3.1 Theorie

Im Theorieabschnitt wird eine Einführung in das Thema Lizensierung gegeben. Es wird veranschaulicht, wie und aus welcher Notwendigkeit heraus diverse Lizensierungsvarianten entwickelt wurden. In weiterer Folge werden unterschiedliche Lizensierungsarten vorgestellt und deren Vor- und Nachteile verdeutlicht. Danach werden die Lizensierungssysteme von

direkten Mitbewerbern der Firma AVL ermittelt, um eine Vorstellung davon zu erhalten, wie die Situation am Markt ist. Abschließend werden noch weitere Lizensierungsarten vorgestellt, die nicht von direkten Mitbewerbern stammen, aber eine Vorstellung vermitteln sollen, welche weitere Alternativen zur Verfügung stehen.

1.3.2 Analyse der AVL Daten

Dieser Teil der Arbeit versucht anhand von realen Daten von der Firma AVL eine Metrik zu finden, die hilft ein Token Based Lizensierungssystem für die angebotenen Softwareprodukte zu schaffen. Hierfür wurden Logfiles von der Firma bereit gestellt, die in weiterer Folge geparsed wurden. Aus den daraus erhaltenen Daten wurden Anwendungs- und Monte Carlo Simulationen durchgeführt. Diese Simulationen hatten zum Ziel, eine Bewertung der Features zu finden, die für unterschiedliche Nutzergruppen anwendbar ist.

1.3.3 Token Based Lizensierung Experiment

Dieses Experiment hat zum Ziel eine Bewertungsbasis in Form von Lookup-Tabellen zu schaffen. Der theoretische Ansatz dieses Abschnitts ist, anhand bekannter Einflussgrößen von Features, die miteinander in einem Tokenpool sind, eine Tokenbewertung durchzuführen. Zuerst wurden diese Einflussgrößen ermittelt bzw. definiert, um dann in weiterer Folge in einem Experiment gegenüber gestellt zu werden. Es wurden diverse Kombinationsmöglichkeiten beleuchtet und die Ergebnisse bewertet. Die Ergebnisse wurden in Lookup-Tabellen eingetragen.

1.3.4 Ergebnisse / Konzept zur Anwendung

Die Lookup-Tabellen, die im vorangegangenen Abschnitt ermittelt wurden, können verwendet werden, um verschiedene Softwareprodukte, die miteinander in einem Token Based Lizensierungssystem sind, zu bewerten. In diesem Arbeitsschritt werden diese Tabellen präsentiert und Beispiele gegeben, wie diese angewendet werden können.
Diese vier Arbeitsschritte werden nochmals im Detail in Abbildung 1.2 dargestellt.

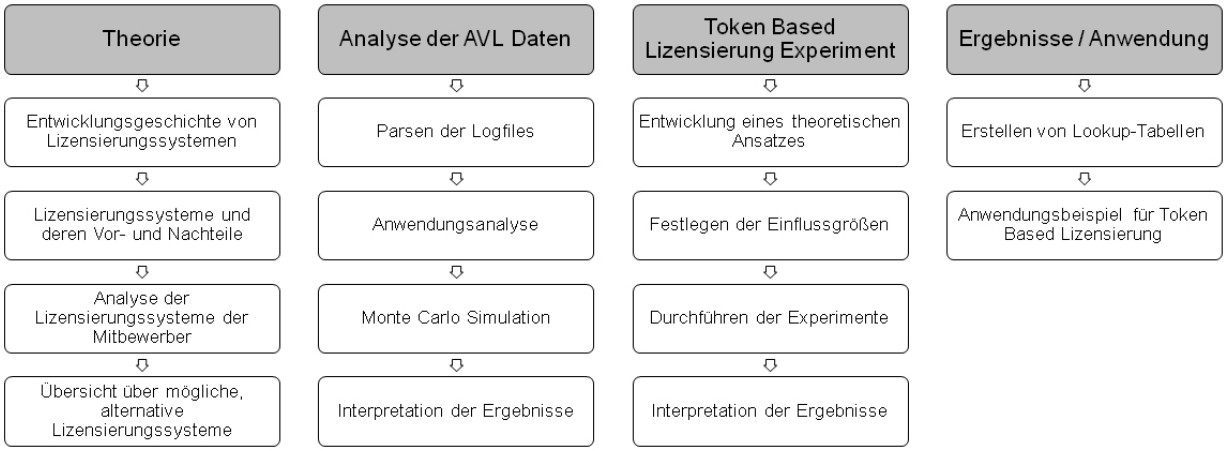

Abbildung 1.2: Detailierte Darstellung der Vorgehensweise innerhalb der vier Arbeitschritte

Kapitel 2

Grundlagen der Softwarelizensierung

In diesem Kapitel wird das Thema Softwarelizensierung aufgearbeitet. Es wird auf einer allgemeinen Ebene auf Herkunft, Zweck und Beweggründe eingegangen (siehe Abschnitt 2.1). Die daraus resultierenden Fakten sollen sicher stellen, dass das Lizensierungssystem optimal seine Aufgabe als Mechanismus zur Verteilung sowie als Werkzeug zur Steuerung des Gebrauchs der Software erfüllen kann. Darüber hinaus werden die Schlüsselfaktoren beschrieben, die zur Findung eines Lizensierungssystems herangezogen werden können (siehe Abschnitt 2.2). Aus diesen Schlüsselfaktoren leiten sich verschiedenste Klassifizierungsverfahren (siehe Abschnitt 2.3) ab, die in weiterer Folge besprochen werden.

2.1 Definition von Software Lizenzen und deren Entwicklungsgeschichte

Im rechtlichen Sinn ist Software als ein Schriftstück zu verstehen. Die Software ist das Intellectual Propterty (IP) des Lizenzhalters. Ihm gehören alle Rechte, die damit verbunden sind (IPR). Dies umfasst laut [Idr04] das Recht auf Benutzung, Veränderung und Verkauf. Eine Lizenz stellt nun einen Konsens zwischen Lizenzgeber und Lizenznehmer dar, der die Benutzung des Softwareprodukts regelt. [TKK09] [TJ05].

Auf dem Softwaremarkt gibt es mehrere Wege um sein IPR zu schützen [DG]. Eine der häufigsten Wege dies zu tun, ist das sogenannte End User License Agreement, oder kurz EULA. Die EULA ist eine Art Vertrag, den der Endnutzer mitsamt der angegebenen Bedingungen zur Nutzung der Software akzeptiert, bevor er Zugang zur Software hat [AB07]. Diese Methode ist sehr häufig im privaten Bereich angesiedelt. Da diese Arbeit

sich jedoch auf den industriellen Bereich spezialisiert, wird auf die EULA nicht weiter ins Detail eingegangen.

Im industriellen Bereich ist die bewährte Methode für Lizensierung eines Softwareprodukts die digitale Lizensierungs-Automatisierung. Bei dieser Methode benutzt der Anbieter ein Lizensierungsmodell, das die Anzahl, Dauer und Häufigkeit der Benutzung durch den Kunden in einer Lizenz regelt. Es existieren verschiedenste Arten von Lizensierungsalgorithmen, die dabei angewandt werden (siehe 2.5), um unterschiedlichste Ausprägungen von Lizenzmodellen zu unterstützen. Diese Lizenzmodelle werden nun durch einzelne Schlüsselfaktoren ermittelt (siehe 2.2). Diese Schlüsselfaktoren variieren von Anbieter zu Anbieter und spiegeln Art, Zweck, Einzigartigkeit der Software und Unternehmensphilosophie wider [Csi09].

Die ersten Arten von Softwarelizensierung entstanden in den 1960er Jahren. Damals war der Markt für Software noch klein. Im Allgemeinen boten verschiedene Anbieter nur ein Softwareprogramm an. Aus diesem Grund war es nicht nötig, sehr komplexe Lizensierungsverfahren einzuführen. Mit dem darauf folgenden Wachstum in der Computerbranche wuchsen die Anbieter für Software mit. Ihr Angebot beschränkte sich nicht nur mehr auf ein einzelnes Softwareprogramm, sondern auf mehrere, eventuell artverwandte oder spezialisierte Programme. Mit dem erweiterten Angebot wurde der Ruf nach anspruchsvolleren Softwarelizensierungsverfahren beim Kunden laut, um Synergien im Softwareportfolio eines Anbieters besser nutzen zu können [Csi09]. Auf Anbieterseite waren die Vorteile eines ausgefeilteren Lizensierungsmodells ebenfalls nicht von der Hand zu weisen. So kann ein attraktiveres Lizenzmodell die Kundenbindung steigern und ebenfalls gegenüber Mitbewerbern einen Vorteil bringen [CTW98]. Die Anforderungen, die auf Anbieter und Kundenseite entstehen, werden in den folgenden Abschnitten noch ausführlich besprochen.

2.2 Schlüsselfaktoren in der Softwarelizensierung

In diesem Abschnitt wird eine theoretische Einführung in die Einflussfaktoren, die zwischen Lizenznehmer und Lizenzgeber wirken, gegeben. Diese Einflussfaktoren bestimmen, welches Lizenzmodell herangezogen wird. Hierbei steht an erster Stelle, wie die Verhandlungen zwischen Kunden und Anbieter verlaufen [SWZ07]. Nach [Idr04] können folgende Schlüsselfaktoren identifiziert werden:

- Domain,
- Scope of Rights,
- Finanzielle Faktoren,
- Zusätzliche Dienste.

In den folgenden Abschnitten werden diese Schlüsselfaktoren weiter im Detail beschrieben.

2.2.1 Domain

Dieser Aspekt bei den Schlüsselfaktoren beschäftigt sich damit, um welche Art von Software es sich handelt. Dabei spielt der Markt, auf dem sich das Produkt befindet, eine große Rolle. Gibt es viele oder wenige Mitbewerber? Ist der Bedarf hoch oder niedrig? Bietet die Software etwas Einmaliges? Dies sind alles relevante Fragen, bei der Gestaltung des Lizenzsystems [DG]. Zum Beispiel, nach [DP09], spiegelt ein Produkt Jahre an Entwicklungsarbeit und Know How wider. Ist dieses am Markt einzigartig bzw. sticht durch besondere Qualität hervor, hat der Lizenzhalter ein starkes Interesse, sein IP zu schützen. Deswegen wird dieser, in der Regel, ein sehr restriktives System bevorzugen. Ist das Produkt hingegen eines von vielen, der Markt also bereits gesättigt, wird der Anbieter ein System wollen, das noch zusätzliche Anreize für den Kunden bietet.

2.2.2 Scope of Rights

Laut [Idr04], ist Scope of Rights der rechtliche Aspekt bei der Lizenzverhandlung. Es wird hier festgestellt, welche Rechte der Kunde vom Lizenzgeber erwirbt. Hier kann es sich um folgende Rechte handeln: abbilden, vorführen, verändern, Derivate erzeugen, benutzen, weiterverkaufen, verteilen oder weiterlizensieren. Dies ist, in einem industriellen Umfeld, ein sehr wichtiger Punkt.

2.2.3 Finanzielle Faktoren

Der Finanzaspekt ist stark abhängig von den beiden vorangegangen Schlüsselfaktoren: Domain und Scope of rights. Diese beiden Faktoren geben den Rahmen vor, in dem sich die wirtschaftlichen Möglichkeiten befinden. Hier geht es vornehmlich um Zahlungsmethoden,

Zahlungszeitraum und Höhe des Zahlungsbetrags [Mac03]. Laut [Idr04] wird in diesen Aspekt eine Menge Arbeit investiert. Eine exakte Anpassung eines Finanzierungssystems für ein Softwareprodukt ist erstrebenswert und kann viel Geld einsparen [DB10]. Hat man zu wenige Lizenzen, müssen Mitwarbeiter warten bis diese frei werden. Dies kann mit einem Produktionsausfall verglichen werden. Bei zu vielen Lizenzen liegen diese Ressourcen brach und man hätte die Kosten dafür anders investieren können.

2.2.4 Zusätzliche Dienste

Dieser Faktor bestimmt, welche weiteren Dienste, zusätzlich zur angebotenen Software noch mit einer erworbenen Lizenz erhältlich sind. Dieser Aspekt kommt zum Tragen, wenn ein Anbieter sich durch zusätzliche Anreize von der Konkurrenz abheben möchte. Dies können zusätzliche Features bei Erreichen einer bestimmten Stückzahl sein, gratis Wartung und Updates, eine Kundenhotline oder Ausbildungseinheiten für Mitarbeiter an der Software.

2.3 Klassifikationsmodelle für Lizenzmodelle

In diesem Abschnitt werden einzelne Klassifikationsmodelle, die für verschiedenste Linzenzmodellarten verwendet werden, vorgestellt. Klassifikationsmodelle werden definiert, um einzelne Lizenzmodelle anhand der zuvor beschriebenen Schlüsselfaktoren (siehe Abschnitt 2.2) einzustufen.

2.3.1 Klassifikation nach Zeit

Bei diesem Modell wird die Errechnung der Zahlungsmodalitäten nach Zeit vorgenommen [KP05]. Der Zeitfaktor regelt die Anzahl, Höhe und Häufigkeit von Einzelzahlungen an den Verkäufer. Diese sind laut [Lic11b]:

- Abonnement: Diese Einteilung fordert einen bestimmten Geldbetrag vom Kunden, um die entsprechende Software für einen bestimmten Zeitraum benutzen zu können. Dieser Zeitraum entspricht für gewöhnlich einem Jahr. Ist die Zeit verstrichen, so endet die Lizenz, und das Programm muss wieder von Neuem erworben werden.

- Unbefristet: Eine unbefristete Lizenz kann vom Kunden frei verwendet werden. Sie ist zeitlich nicht befristet. Dies ist jedoch meist auf diese Version der Software be-

schränkt. Will ein Kunde eine aktuellere Version, so muss er die neuere Software kaufen, oder, falls möglich, ein Upgrade erwerben.

2.3.2 Klassifikation nach technischen Kriterien

Diese Klassifikation versucht die einzelnen Lizensierungsarten nach Art des Zugangs zur Software zu untergliedern. Zugang ist in diesem Zusammenhang als physikalisch zu betrachten. Die lizenzierte Software ist auf Hardware zugeschnitten, wie zum Beispiel, Einzelcomputer oder Netzwerke.

Dieses Klassifikationsmodell unterscheidet folgende drei Ausprägungen [Mac06] [KP05]:

- Feste Lizenzen (auch bekannt als Node-locked oder CRC Lizenz)
- Freie Lizenzen
- Hardware-locked Lizenzen

Feste Lizenzen

Eine feste Lizenz bzw. Node-locked Lizenz ist nur auf einem einzelnen Computer gültig [LZWM08]. Bei Installation wird ein einzigartiger Schlüssel für den jeweiligen Computer generiert, der benutzt wird, um die Lizenz zu erstellen. Dadurch ist diese Software an die Maschine gebunden, und kann zwar auf anderen Computern installiert, aber nicht ausgeführt werden.

Freie Lizenzen

Freie Lizenzen bzw. 'floating' Lizenzen werden für einen Server, innerhalb eines Netzwerks, ausgestellt. Dieser Server kann Computern, die diesem Netzwerk angehören, Lizenzen zuteilen, wenn diese benötigt werden. Dies bedeutet effektiv, dass die Software auf mehreren Computern installiert sein kann, die Anzahl der Lizenzen, die gleichzeitig im Netzwerk verfügbar sind, können aber nicht einen vordefinierten Wert übersteigen. Wird die Software nicht mehr benutzt, wird die ausgestellte Lizenz wieder an den Server zurück gegeben. Diese Art der Lizensierung wird oftmals in Firmen angewandt, um weniger häufig benutzte Software optimal nutzen zu können [Mac06].

Hardware-locked Lizenzen

Die Hardware-locked Lizenz arbeitet, wie die feste Lizenz, mit einem Schlüssel. Der Unterschied ist jedoch, dass der Schlüssel, der die Benutzung auf eine Maschine festlegt, nicht dynamisch generiert wird. Hier wird der Schlüssel auf einem Hardwarespeichermedium geladen. Üblicherweise ist dies ein USB Speicher, der dann manuell angeschlossen wird. Dies ermöglicht die Nutzung der Software auf mehreren Maschinen und gleichzeitig die Regelung der Anzahl der gleichzeitig laufenden Instanzen der Software. Eine praktische Anwendung wäre zum Beispiel für jemanden, der auf Geschäftsreisen auf einem mobilen Gerät arbeiten muss, am Arbeitsplatz jedoch nicht [KP05].

2.4 Entstehung von Softwarelizensierungsmodellen

Die Entwicklung unterschiedlicher Lizenzmodelle wurde vor allem durch das Vertrauen zwischen Kunden und Verkäufer sowie durch neue Technologien geprägt [Lic11a]. Mit Vertrauen wird die tiefe Beziehung bemessen, die Lizenzgeber und Lizenznehmer bereit sind miteinander einzugehen. Ist ein Verkäufer nicht davon überzeugt, dass ein Kunde weiterhin in das Produkt investiert, ist das Verhältnis eher auf kurzfristigen Gewinn ausgelegt. Glaubt der Anbieter, dass ein Kunde langfristig investiert, so ist man auch bereit dem Kunden mehr Freiheiten zu gewähren, um die Bindung zu stärken [Csi09].

Laut [Lic11b], gab es am Anfang die traditionellen Kunde-Verkäufer Modelle. Diese umfassten Einzel-, geteilte und zeitraumbezogene Lizenzen. Hat ein Kunde nur ein einziges Produkt eines Verkäufers lizensiert, so wird dies als Einzellizenz bezeichnet. Diese Einzellizenz ist meist ebenfalls auf einen einzigen Computer beschränkt. Eine geteilte Lizenz wurde notwendig durch die aufkommende Netzwerktechnologie in den 80er Jahren. Hierbei ist es möglich, auf eine Lizenz, unabhängig vom Computer oder User, im Netzwerk zuzugreifen. Eine weitere Art der Lizensierung ist die Zeitraumbezogene Lizenz. Sie ist auch als 'Demo Lizenz' bekannt. Sie ermöglicht den Zugang zur Software nach der Installation für einen beschränkten Zeitraum. Diese Art der Lizensierung erfüllt drei verschiedene Aufgaben. Zu aller erst ermöglicht sie dem Kunden die Software zu testen und evaluieren. Weiters kann sie bei Kapazitätsengpässen (mehr Benutzer als Softwarelizenzen vorhanden) aushelfen und ihr Einsatz kann große Ausgaben für eine Firma für einen kurzen Zeitraum aufschieben.

Der nächste Schritt in der Evolution der Softwarelizenzen sind die Klassifikationsmodelle. Sie sind im Grunde eine Erweiterung der Einzellizenzen. Nach diesem Modell ist

es dem Benutzer erlaubt bestimmte Features einer Software zu nutzen, je nachdem, wie ihm zuvor die Rechte dazu zugeteilt wurden. Es werden unterschiedliche Rechtekategorien vordefiniert und Benutzer zugeteilt (z.B. Admin, User). Diese Rechte müssen nicht zwangsläufig auf den Umfang des Zugangs zur Software beschränkt sein, es können auch Regeln für Zeiträume der Nutzung definiert werden [Lic11b].

Ein weiterer Schritt in der Evolution der Softwarelizensierung sind die Pay-Per-Use Modelle. Hier wird nur der tatsächliche Gebrauch erfasst, und je nach Umfang die Kosten errechnet. Diese Modelle sind für den Kunden die effektivsten, da es keine 'Leerlauf' Zeit gibt, die der Kunde bezahlen muss ohne Nutzen davon zu tragen. Bei diesem System treten allerdings zwei Probleme auf [Lic11a]:

- Wie kann der Verkäufer den tatsächlichen Gebrauch messen, um danach seine Rechnung zu erstellen?
- Wie kann der Kunde jederzeit seine Kosten für die Software überprüfen, um sicher zu stellen, dass das Budget nicht überschritten wird?

Oftmals werden diese Probleme durch Logfiles gelöst. Diese speichern die Gebrauchsdaten der Lizenzen über eine bestimmten Zeitraum hinweg, um im Anschluss zum Verkäufer gesendet und analysiert zu werden [Lic11b].

Remix Modelle sind eine Mischung aus den zuvor beschriebenen Modellen mit einem hohen Grad an Flexibilität für den Kunden. Diese wurden eingeführt, um den Bedürfnissen der Verkäufer, Kunden und auch der Software besser gerecht werden zu können [Lic11b]. Es wird nicht eine bestimmte Anzahl an Lizenzen an den Kunden vertrieben, sondern diese sind frei zugänglich. Mit diesem Modell versucht der Verkäufer im Normalfall, die Kundenbeziehung zu stärken und ist nicht auf kurzfristigen Gewinn, sondern auf eine langfristige Partnerschaft aus. Diese Remix Modelle können in zwei Arten unterschieden werden.

- Static Remixing: Hierbei werden in regelmäßigen Abständen die benutzbaren Lizenzen neu ausgewählt. Der Kunde kann sich in Abständen von 3-6 Monaten, je nach Bedarf, einen neuen Lizenzmix aussuchen. Dies ist in großem Maße nützlich, wenn man eine genaue Vorstellung hat, was in der kommenden Periode benötigt wird.
- Continuous Real-time Remixing: Ist auch als Token Based Licensing bekannt. Hier wird jedem Produkt ein relativer Tokenwert zugewiesen, und der Kunde kann sich

einen Pool an verwendbaren Token erwerben, die dann dynamisch verbraucht werden. Auf diese Art der Lizensierung wird später in Abschnitt 2.5.6 detailiert eingegangen.

Ist das Angebot an unterschiedlichen Lizensierungsarten für ein Produkt sehr groß, wird auf Kundenseite oftmals auf License Portfolio Management (LPM) zurückgegriffen [GW09]. LPM ist eine Optimierung der eingekauften Lizenzen an den Bedarf.

Technology Partnership ist die engste Form der Zusammenarbeit zwischen den Parteien. Hier ist das höchste Maß an Vertrauen notwendig, da das Softwarelizenzmodell hier auf dem Fair Use Prinzip aufgebaut ist. Hier spricht man nicht mehr von einer Kunde-Verkäufer-Beziehung, da diese Zusammenarbeit eher zwischen Firmen stattfindet und nicht dem Gewinn, sondern dem Bereitstellen von Ressourcen und Know How dient [Lic11b].

Abbildung 2.1 zeigt wie die einzelnen Lizenzmodelle auf einander aufbauen.

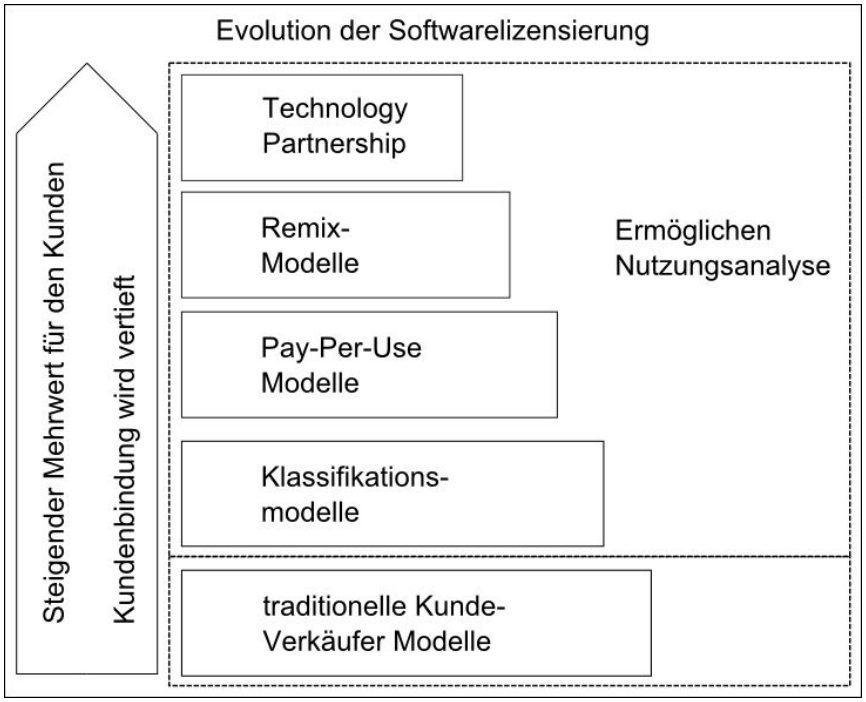

Abbildung 2.1: Evolution von Softwarelizenzen (adaptiert von [Lic11b])

2.5 Vergleich von Softwarelizensierungsmodellen

Nachdem im letzten Abschnitt die Entwicklung der Softwarelizensierung beschrieben wurde (siehe 2.4), werden in den folgenden Abschnitten einzelne Lizensierungsmodelle genauer beschrieben.

2.5.1 Single User Lizenz

Eine Single User Lizenz ist die einfachste Variante der Lizensierung. Dieses Lizenzmodell sieht vor, dass ein einzelnes Softwareprodukt einem einzelnen Nutzer zugänglich gemacht wird. Auch wenn diese Art der Lizensierung einfach erscheint, gibt es dennoch mehrere Faktoren die beachtet werden müssen. Die Faktoren, die diese Art der Lizensierung einschränken, sind Mensch und Zeit. Single User Lizenzen sind in ihrer Verwendung für einen bestimmten Zeitraum auf einen User beschränkt [Mac06].

2.5.2 Netzwerklizenz

Die Netzwerklizenz ist ein weiteres Lizensierungsmodell. Die Netzwerklizenz funktioniert innerhalb eines Netzwerks oder einer Domain. Jeder Computer innerhalb des Netzwerks hat Zugriff auf die Lizenz. Im Gegensatz zur Single User Lizenz ist der limitierende Faktor nicht der Mensch, sondern die Anzahl der verfügbaren Lizenzen innerhalb des Netzwerks. Die Anzahl der simultan verwendeten Lizenzen wird von einem License Management System verwaltet. Zeit wird ebenfalls als limitierender Faktor angesehen, da die Nutzung von Netzwerklizenzen, wie auch Single User Lizenzen, an einen Zeitrahmen gebunden ist [Mac06].

2.5.3 On-Demand Lizenz

Die On-Demand Lizenz ist eine wesentlich dynamischere Art der Lizensierung. Hier gibt es keine limitierenden Faktoren betreffend der Nutzung der Software. Dem Lizenznutzer steht die Software zur freien Verfügung, unabhängig von Zeit oder Anzahl der simultan genutzten Instanzen. Die Vorteile eines solchen Lizensierungsmodells sind folgende [GW09]:

- Keine Engpässe. Da es in der Nutzung der Software keine limitierenden Faktoren gibt, kann jederzeit auf einen größeren Bedarf an Lizenzen eingegangen werden.
- Kein Leerlauf. In diesem Lizensierungsmodell wird nur die tatsächliche Nutzungszeit als Bewertungsgrundlage für den Preis herangezogen. Lizenzen, die über eine Zeitperiode abgerechnet werden, verursachen auch Kosten, wenn sie nicht in Benutzung sind.
- Keine Vorauszahlung. Da die Nutzung der Software erst am Ende der Abrechnungsperiode ermittelt werden kann, kann auch erst dann ein Zahlungsbetrag anfallen.

Netzwerklizenzen und Single User Lizenzen werden oftmals im Voraus bezahlt. Dieser finanzielle Aspekt kann auch eine entscheidende Rolle bei der Kaufentscheidung sein.

Diese Vorteile werden von folgenden Nachteilen begleitet [Mac03]:

- schwierige Kostenplanung. Da die Kosten für die Softwarelizenzen nicht exakt vorab berechnet werden können, sind diese schwierig zu budgetieren. Viele Firmen bevorzugen ein Modell, bei dem Kosten vorab bekannt sind.
- höherer Preis. Mit höherer Dynamik des Lizensierungsmodells steigt der Preis. Dem Kunden wird nicht nur die Leistung, die durch die Software zur Verfügung gestellt wird, berechnet, sondern auch der Freiheitsgrad unterschiedliche Software zu benutzen. Je höher der Freiheitsgrad, desto höher der Preis für die tatsächliche Nutzung.

On-Demand Lizensierung ist in der Industrie eher selten anzutreffen. Die Zusatzkosten, die durch den erhöhten Freiheitsgrad anfallen, und die Probleme ein solches Lizensierungsmodell zu budgetieren, überwiegen oftmals die Vorteile.

2.5.4 Prozessor Lizenz

Die Prozessor Lizenz ist eine eher neuartige Art der Lizensierung. Sie richtet sich hauptsächlich nach technischen Entwicklungen der letzten Jahre. Leistungsstarke Multi Core Prozessoren sind, dank des technischen Fortschritts der letzten Jahre, allgemein verbreitet und erschwinglich. Mit dem Gebrauch von Multi Core Prozessoren können Berechnungen, die früher viel Zeit in Anspruch nahmen, erheblich beschleunigt werden. Unterstützt die lizensierte Software Berechnungen mittels Mehrkernprozessor, berechnet der Lizenzgeber einen Aufschlag pro Rechenkern, da der Kunde mit erheblicher Zeit und Ressourcenersparnis rechnen kann. Dies ist, zum Beispiel, ein Fixbetrag für die Grundlizenz plus 20 % Aufschlag pro Rechenkern. Oftmals wird der Aufschlag ebenfalls reduziert mit steigender CPU Anzahl. Dies wird angewandt, da zwei Rechenkerne nicht gleich zwei Lizenzen auf zwei separaten Maschinen entsprechen. [GW09]

2.5.5 Server Lizenz

Eine Server Lizenz ist von der Funktion her mit einer Netzwerklizenz vergleichbar. Innerhalb eines Netzwerks können Benutzer auf die Lizenz zugreifen. Es wird jedoch nicht nach

Anzahl der Benutzer beschränkt, wie bei der Netzwerklizenz, sondern ist auf den Server lizensiert. Die Benutzer nutzen die Lizenz nicht, sondern der Server, der seine Funktionen als Service bereit stellt [GD06].

OEM Lizenz

OEM steht für 'Original Equipment Manufacturer' bzw. Erstausrüster. Die OEM Lizenz bezeichnet meist die lizensierte Software, die bereits bei Erwerb von Hardware auf einer Maschine installiert oder mitgeliefert ist. Dies hat den Hintergrund, dass sich Hardwarehersteller oft wegen Garantie- und Gewährleistungsvorschriften absichern möchten und diese Software empfehlen bzw. testen. Dies geschieht, zum Beispiel, oft beim Erwerb von Brennern. Eine weitere Variante ist Prestige oder Marketing. Ein gutes Beispiel hierfür wäre Microsoft[1]. Windows ist oft mit einer verbilligten Lizenz bei gekauften PC's und Laptops vorinstalliert. Diese Vorgehensweise sichert eine weite Verbreitung für das Microsoft Betriebssystem.

2.5.6 Token Based Lizensierung

In diesem Abschnitt wird Token Based Lizensierung behandelt (siehe Definition IBM[2]). Diese Art der Lizensierung ist ein Produkt aus mehreren unterschiedlichen Lizensierungsarten und somit ein Remix Modell (siehe Abschnitt 2.4). Es wird dabei versucht die Vorteile von Netzwerklizenzen und On-Demand Lizenzen zu vereinen und dabei, wenn möglich, die Nachteile zu minimieren. Als Grundprinzip gilt die Annahme, dass unterschiedliche Software eines Anbieters angeboten wird. Diese Software wird in einem Paket angeboten. Innerhalb dieser Pakete werden artverwandte Softwareprodukte platziert. Diese Produkte werden nicht einzeln lizensiert, sondern ihnen wird ein Wert beigemessen, der dem relativen Wert zu allen anderen Produkten im Paket entspricht. Dieser Wert ist der Tokenwert des Produkts. Um ein Produkt benutzen zu können, erwirbt der Kunde nun eine beliebige Anzahl an Token. Diese Token sind der Tokenpool und können als verfügbare Währung gesehen werden, um die Software zu benutzen. Checkt man ein Produkt aus, so verringert sich der Tokenpool um diesen Wert. Danach wird nach zwei möglichen Varianten verfahren. Die benutzten Token verfallen bei einem konsumptiven Algorithmus [KP05], oder sie werden bei einem permanenten Algorithmus an den Tokenpool zurück gegeben.

[1] www.microsoft.com

[2] https://licensing.subscribenet.com/control/ibmr/faqs#5-3b

In Abbildung 2.2 wird der permanente Algorithmus beispielhaft dargestellt. Es wird ein Tokenpool von 5 angenommen mit zwei Features A und B. Feature A besitzt einen Wert von 3, während Feature B einen Wert von 2 besitzt. Es können nun maximal Features im Wert von 5 Token zur gleichen Zeit ausgecheckt sein. Aus dieser Annahme lassen sich nun folgende Möglichkeiten ableiten.

- 1 x Feature A (3)
- 1 x Feature B (2)
- 2 x Feature B (4)
- 1 x Feature A und 1 x Feature B (5)

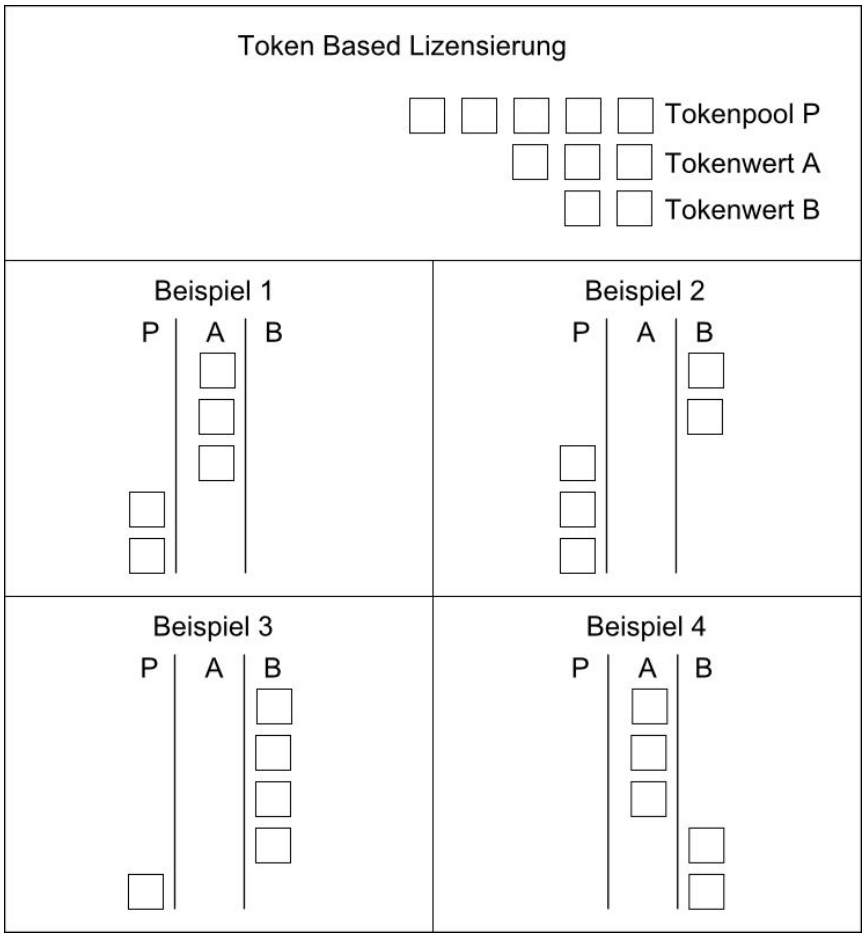

Abbildung 2.2: Darstellung des permanenten Token Based Algorithmus für einen Tokenpool von 5 Token

Die Vorteile eines solchen Verfahrens sind:

- einfache Budgetierung,
- dynamische Lizenzbenutzung nach Bedarf,
- höhere Kundenbindung und
- kann als Anreizsystem dienen.

Es wird der Tokenpool zu Anfang der Abrechenperiode erworben. Dies gewährleistet dem Kunden eine einfache Vorausplanung der Kosten ohne unerwartete Zusatzkosten. Es bleibt dem Kunden dennoch die Freiheit individuell zu entscheiden, wann er wie viele Lizenzen nutzen möchte, solange die Gesamtwertigkeit kleiner gleich dem Tokenpool bleibt. Man könnte dies mit der On-Demand Lizensierung ohne deren Nachteile vergleichen. Für den Anbieter eines solchen Lizensierungsverfahrens kann ein dynamisches System neue Kunden anlocken und bestehende festigen. Der Kunde wird angehalten, zur Verfügung stehende Token zu nutzen, auch wenn zuvor diese Software nicht gebraucht wurde. Dadurch entsteht im optimalen Fall neuer Bedarf, der durch eine Erhöhung des Tokenpools gedeckt werden muss. In weiterer Folge wird die Bindung des Kunden größer, desto mehr Software vom selben Anbieter genutzt wird, und nicht auf Konkurrenten zurück gegriffen wird. Die Nachteile, die ein solches System hat sind folgende:

- Formen von Paketen,
- Bewertung der Features und
- Möglichkeiten zum Austricksen des Systems.

Um ein Token Based Lizensierungmodell einzurichten, sind sehr viele Denk- und Rechenschritte nötig. Es müssen artverwandte Features ausgewählt werden, die in Relation zueinander stehen, um bewertet werden zu können. Die Bewertung der Features ist der nächste Schritt. Mit jedem zusätzlichen Feature in einem Paket steigt die Komplexität. Wie man in Abbildung 2.2 sieht, können bereits wenige Features oder ein großer Tokenpool eine hohe Anzahl an Möglichkeiten aufweisen. Eine falsche Bewertung kann zur Folge haben, dass Features so billig werden, dass sie praktisch 'gratis' sind, und andere zu teuer. Dass ein solches System leicht täuschbar ist, ist ebenfalls eine große Herausforderung beim Entwurf eines Tokensystems. Es existieren sogenannte Lizenzoptimierer, die weniger wichtige Tasks auf Nachtstunden verschieben, um Token für den Tag frei zu halten. Dies führt

dazu, dass der Ressourcenverbrauch auf den gesamten Tag gleichmäßig verteilt ist, und es zu geringerer Auslastung kommt.

2.6 Zusammenfassung und Fazit

In diesem Kapitel wurde auf Lizensierung von Software im Allgemeinen eingegangen. Es wurde die Entstehungsgeschichte für unterschiedliche Arten der Lizensierung erläutert und welche Vor- und Nachteile diese mit sich bringen. Eine Art der Lizensierung, die noch sehr jung ist, aber dennoch rasch an Popularität gewann, ist die Token Based Lizensierung. Dieses Verfahren soll eine Mischung aus traditionellen festen Lizenzen und einem dynamischen On-Demand Verfahren bilden und deren Vorteile, individuelle Softwarenutzung und Kostenplanung, in sich vereinen. Die Nachteile eines solchen Systems liegen in der Bewertung der Features, die darin zur Verfügung gestellt werden. In den Kapiteln 4, 5, 6 und 7 wird geschildert, wie im Laufe dieses Projekts ein Ansatz gesucht wurde, um die in Abschnitt 2.5.6 aufgeworfene Herausforderung der Bewertung von Features und die Umsetzung eines Tokensystems durchzuführen.

Kapitel 3

Lizensierungsverfahren der Mitbewerber von AVL

In diesem Kapitel der Arbeit wird auf die Mitbewerber der Firma AVL genauer eingegangen. Es wurde durch Recherche ermittelt, welche Lizensierungsverfahren die direkten Mitbewerber anwenden und ob auf Grund dessen Handlungsbedarf für die AVL besteht. Die untersuchten Mitbewerber sind:

- LMS
- ANSYS
- MSC

In weiterer Folge wurden noch weitere Lizensierungssysteme in anderen Fachbereichen untersucht, um einen Überblick über mögliche Alternativen bzw. Anregungen für eine individuelle Lösung zu erhalten.

3.1 Lizensierungsverfahren: LMS

LMS ist ein Anbieter für Test und Mechatronik Simulationssoftware im Fahrzeug-, Luftfahrt- und industriellen Fertigungsbereich[1]. Über die Website der Firma lassen sich keine Rückschlüsse auf das Lizensierungsverfahren ziehen. Aus diesem Grund wurden Lizenzfiles der TUGraz eingesehen. Diese Lizenzfiles zeigen (siehe Abbildung 3.1), dass LMS mit einfachen Jahreslizenzen arbeitet.

[1] http://www.lmsintl.com

```
FEATURE lms_spectralacq LMS_INTL 6.0 30-dec-2006 1 XXX\
HOSTID=XXX BORROW=672ISSUER="LMSInternational"\
ISSUED=12-Jan-2006 NOTICE="Licensed to: Technische UNIVERSITAET\
GRAZ UNI TU GRAZ" START=12-Jan-2006 SIGN=XXX
```

Abbildung 3.1: Ausschnitt aus einem LMS Lizenzfile der TUGraz

3.2 Lizensierungsverfahren: ANSYS

ANSYS arbeitet im Bereich der 'Methode der finiten Elemente', 'Numerische Strömungsmechanik', Elektronik und Elektromagnetik sowie der Design Optimierung[2]. Für die Produkte dieser Firma wurden ebenfalls Lizenzfiles der TUGraz eingesehen, um heraus zu finden, welche Lizenzen hier angeboten werden. Dabei wurden 2 verschiedene Typen identifiziert:

- permanente Lizenz (siehe Abbildung 3.2)
- periodische Lizenz (siehe Abbildung 3.3)

```
INCREMENT agppi ansyslmd 1998.1215 permanent 3 XXX \
VENDOR_STRING="customer:XXX timezones:0,1,2,23" SUPERSEDE \
ISSUED=28-jan-2010 START=23-jan-2010
INCREMENT piproe ansyslmd 1998.1215 permanent 3 XXX \
VENDOR_STRING="customer:XXX timezones:0,1,2,23" SUPERSEDE \
ISSUED=28-jan-2010 START=23-jan-2010
```

Abbildung 3.2: Ausschnitt aus einem ANSYS Lizenzfile der TUGraz für Lizenztyp 1

```
INCREMENT aa_turbo ansyslmd 9999.9999 08-jun-2010 1 XXX \
VENDOR_STRING="eval customer:XXX" ISSUER=AUFI_UNIV \
ISSUED=27-apr-2010 START=28-apr-2010
```

Abbildung 3.3: Ausschnitt aus einem ANSYS Lizenzfile der TUGraz für Lizenztyp 2

Es besteht hier die Möglichkeit zu entscheiden, ob man die Software komplett, unbegrenzt oder sie nur für ein Jahr lizensiert.

3.3 Lizensierungsverfahren: MSC

MSC Software vertreibt Simulationssoftware im Bereich linearer und non-linearer 'Methode der finiten Elemente', 'Multi-Body-Dynamik', Kontrollsysteme und vieler andere Ap-

[2] http://www.ansys.com/

plikationen. Die Einsichtnahme der Lizenzen der TUGraz brachte folgendes hervor (siehe Abbildung 3.4).

```
FEATURE ADAMS_Tire_FTire MSC 2008.0331 31-dec-2099 50 \
XXX VENDOR_STRING=PID:10229 \
ISSUED=01-jun-2007 ck=252 SN=XXX
FEATURE ADAMS_3DRoad MSC 2008.0331 31-dec-2099 50 \
XXX ISSUED=01-jun-2007 ck=252 SN=XXX \
SN=XXX
```

Abbildung 3.4: Ausschnitt aus einem MSC Lizenzfile der TUGraz

Auch hier ist es möglich eine jahresbasierte Lizenz zu erwerben. Auf der Website der Firma findet sich zusätzlich noch ein Angebot für den Erwerb von 'Enterprise Advantage´s License Units'[3]. Diese 'Units' können als universelle Währung für die gesamte Software in einem vordefinierten Softwarepool verwendet werden. Somit bietet MSC Software als Einziger der untersuchten Mitbewerber bereits Token Based Lizensierung an.

3.4 Beispiele für alternative Lizensierungsverfahren aus anderen Branchen

In diesem Abschnitt werden alternative Lizensierungsverfahren erläutert. Diese Verfahren wurden im Zuge der Recherche zu Lizensierungsverfahren gefunden und wurden nicht von direkten Mitbewerbern der Firma AVL angewandt. Dennoch zeigen sie auf, wie man individuelle Lösungen anstreben kann, die auf die jeweilige Software zugeschnitten ist. Dies ist besonders wichtig, da es in diesem Bereich keine Patentlösung gibt.

3.4.1 Lizensierungsverfahren: Altair Hyperworks

Altair Hyperworks ist kein direkter Mitbewerber der AVL. Er wird jedoch, im Rahmen dieser Arbeit, aufgrund seiner sehr innovativen und kreativen Lizenzsystems näher erläutert. Hyperworks ist eine Plattform für 'Modellierung und Visualisierung', 'Analyse und Optimierung' sowie für 'Enterprise Software'[4]. Das Lizenzfile der TUGraz ist in Abbildung 3.5 dargestellt.

[3] http://www.mscsoftware.com/offers/ea.cfm
[4] http://www.altairhyperworks.com

```
FEATURE GridWorks altair_lm 10.0 31-aug-2010 5000 XXX \
Vendor_info=XXX BORROW \ SN=XXX SIGN2="XXX"
```

Abbildung 3.5: Ausschnitt aus einem Hyperworks Lizenzfile der TUGraz

Dies lässt darauf schließen, dass hier mit Token Based Lizensierung verfahren wird. Eine weitere Untersuchung der Angebotsdokumente der Firma Hyperworks an die TUGraz lässt erkennen, dass es sich hierbei jedoch um ein spezielles, an die Organisation individuell angepasstes, Token Based Lizensierungsschema handelt. Hyperworks bietet seine Software in unterschiedlichen Stufen an, die von 0 bis 5000 GridWorks Units reichen (diese Stufen sind: 0, 3, 400, 600, 650, 2100, 2500, 3800 und 5000, siehe Abbildung 3.6). Benutzt man nun ein Programm einer Stufe, werden die entsprechenden Punkte dafür ausgecheckt. Zusätzlich können aber auch alle anderen Programme, die auf derselben, oder auf einer geringeren Stufe sind, gratis mit benutzt werden, ohne weitere Punkte auszuchecken. Die folgende Darstellung zeigt, welche Programme welcher Stufe zugewiesen sind [5].

	GridWorks Units(GWU)
HyperViewPlayer	0
PBS Professional	3
DataManager	400
HyperGraph	600
HiQube	650
HyperMesh	2100
RADIOSS	2500
MotionSolve	3800
OptiStruct	5000

Abbildung 3.6: Ausschnitt aus der Hyperworks Preistabelle für GridWorks Units

3.4.2 Lizensierungsverfahren: Silvaco

Silvaco bietet Software zur Simulation von analogen oder gemischten Schaltkreisen[6]. Das Interessante an diesem Beispiel ist die Vielfalt an unterschiedlichen Kombinationsmöglichkeiten, die angeboten werden, um einzelne Softwareprodukte und deren Kategorien zu lizensieren. Die folgende Darstellung zeigt eine Landkarte der Produkte (Abbildung 3.7[7]).

Aus dem Angebot der Firma geht hervor, dass es separate Token für das Paket TCAD und das Paket EDA gibt (siehe Abbildung 3.7, [SIL10b] und [SIL10a]). Außerdem gibt es

[5] http://www.altairhyperworks.com/Solutions,1,24,hyperworksondemand.aspx
[6] http://www.silvaco.com/
[7] http://www.silvaco.com/products/index.html

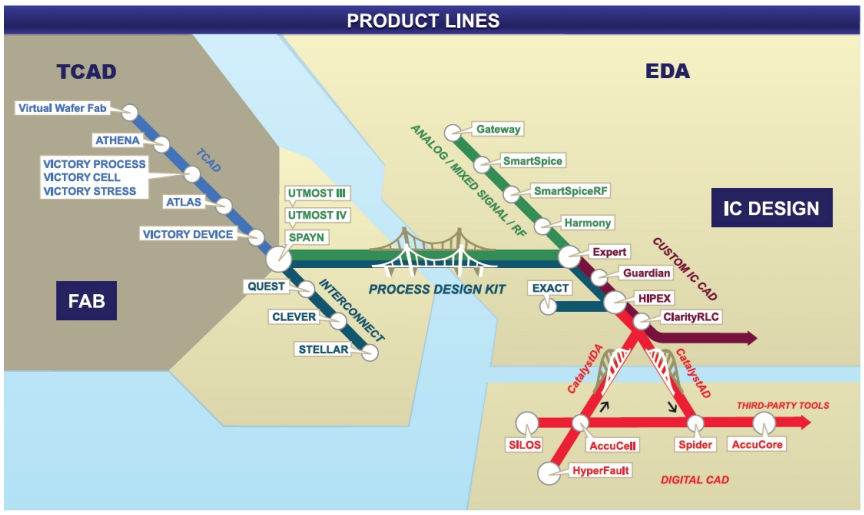

Abbildung 3.7: Produktlandkarte für Silvaco Software (aus Produktkatalog)

noch Kombitoken für beide Pakete [SIL12]. Zusätzlich gibt es noch die Möglichkeit den traditionellen Weg mit Jahreslizenzen zu gehen oder sich permanente Lizenzen zu kaufen. Eine weitere Option die geboten wird, ist die 'Company Unlimited' Lizenz. Diese lassen sich online erwerben und gelten für alle Produkte, erschöpfen sich jedoch, nach einmaligem Gebrauch (in der Regel nach 24 h ungültig).

3.5 Zusammenfassung und Fazit

Aus der Analyse der direkten Mitbewerber ging hervor, dass LMS sowie ANSYS keine Token Based Lizensierung anbieten. Die Preisgestaltung verhielt sich bei deren Produkten ähnlich denen der Firma AVL. D.h. der Kunde kann sich einzelne Features auf Zeitbasis (i.a. jährlich) lizensieren oder für einen Permanentkauf entscheiden. MSC Software hat in seinem Angebot noch zusätzlich das Token Based Lizensierungsmodell. Aus der Analyse weiterer Softwareanbieter und deren Lizenzmodelle geht hervor, dass das Token Lizensierungsmodell nicht nach einem strikten Schema vorgehen muss, und es durchaus Spielraum für kreative und individuelle Lösungen gibt. Diese können eventuell auch einen Bonus gegenüber den Mitbewerbern verschaffen, wenn man diese nach seinen Stärken gestaltet. In [SG04] werden die LANDSCAPE-ALTERING FORCES vorgestellt. Es werden die 6 Hauptfaktoren beschrieben, die einen Betrieb dazu veranlassen, sein Softwarelizensierungsverfahren zu wechseln. Diese sind beschrieben wie folgt:

- Wunsch nach leichter vorhersehbaren Einnahmen durch den Verkäufer
- Mehr Flexibilität und Einfachheit in der Softwarenutzung durch den Kunden
- Subjektive Änderung des Werts der Software (z.B. flexibleres System als Reaktion auf Wertverringerung)
- Verstärktes Interesse an anderen, alternativen Lizensierungssystemen durch den Kunden
- Verkäufer will Akzeptanz und/oder Kundenbindung erhöhen
- Technologische Neuerungen wie z.B. Open Source und Virtualisierung

Auf die Firma AVL treffen mehrere dieser Punkte zu. Nach Beratung und Analyse der in den voran gegangen Abschnitten vorgestellten Ergebnisse, wurde beschlossen, weiter in Richtung Token Based Lizensierungssystem zu forschen.

Kapitel 4

Analyse von Benutzerdaten

Ein für die Firma AVL angepasstes Token Based Lizensierungsmodell sollte anhand von Gebrauchsdaten der Software ermittelt werden. Es sollte ein Weg gefunden werden, das bestehende Lizensierungsverfahren zu analysieren, einzelne Lizenzen zu bewerten, um sie dann mit einander für ein Tokensystem kombinieren zu können. Die Daten für diese Schritte wurden von der AVL zur Verfügung gestellt. Eine interne Preisliste für die angebotenen Features, eine Lizenzdatei wurden herangezogen, um den Aufbau und die Preisverteilung der Software zu analysieren (siehe 4.2), und Serverlogfiles über die Benutzung der Software, um das Auscheckverhalten zu ermitteln (siehe 4.4), wurden bereit gestellt.

4.1 Aufbau des Produktportfolios

Die Firma AVL hat einen Zweig für Simulationssoftware für Motoren: AST - Advance Simulation Technologies. Es werden verschiedenste Softwarelösungen für Simulationssoftware angeboten. Diese unterteilt sich in 4 große Pakete:

- Boost
- Fire
- Cruise
- Excite

Diese 4 Pakete sind nicht komplett getrennt von einander zu behandeln, da sie auch, wegen ihrer Artverwandtheit, auch miteinander in einer sinnvollen Vorgehensweise benutzt werden können.

4.1.1 Boost

AVL Boost ist ein voll integriertes 'virtuelles Motoren Simulations Tool', zur akkuraten Vorhersage über das Verhalten eines Motors. Dies umfasst Leistung, Akustik und die Effektivität der Abgasbehandlung [AVL08]. Das Boost Hauptpaket stellt die grundlegenden Werkzeuge bereit, um einen Motor auf sein thermodynamisches Verhalten zu untersuchen. Dies kann als Basisschritt in der Entwicklung eines Motors gesehen werden. Das Paket beinhaltet ein umfassendes Grafisches Userinterface (GUI), diverse Preprozessoren, um die Erstellung eines Modells zu erleichtern, sowie Postprozessoren, um die Resultate effizient zu vergleichen und analysieren.

'Linear Acoustics' ist ein zusätzliches Paket in der Produktpalette, um einen Motor im Speziellen auf seine akustischen Eigenschaften zu untersuchen. Die 'Linear Acoustics' Komponente ermöglicht die Simulation von Ansaug- und Ausstoßgeräten und eine akkurate Vorhersage von Übertragungs-, Einfügungs- und Lautstärkenverlust [AVL10a].

Die 'Boost Aftertreatment' Komponente dient zur Simulation des gesamten Flüssigkeitsbedarfs, der Wärmeentwicklung und Verteilung und der chemischen Reaktionen innerhalb eines Motors. Dies beinhaltet alle Arten von Gasausstoß und Verteilung sowie katalytische Reaktionen [AVL10b].

4.1.2 Fire

AVL Fire ist eine multifunktionelle Thermo-fluid Software, um Flüssigkeitsdynamik in einem drei dimensionalen Raum darzustellen. Dies beinhaltet ein automatisiertes 3D Grid Tool, das auf hexaedrischen Elementen basiert, und in beliebig granulare Schichten untergliedert werden kann. Die von Fire durchgeführten Simulationen verwenden einen auf Druck basierten Algorithmus. Für Turbulenzberechnungen werden Algorithmen wie $k - \epsilon$ angewandt, die im AVLs Hybrid Turbulence Model (HTM) oder im Reynolds Stress Model (RSM) verwirklicht sind [AVL09].

Zusätzliche Features sind:

- AVL Code Coupling Interface (ACCI),
- Conjugate heat transfer,
- General gas phase reactions,
- Porosity module,

- Radiation,
- Single phase boiling,
- Species transport,
- Thin walls und
- User-defined functions.

4.1.3 Cruise

AVL Cruise ist ein Simulationstool zur Fahrzeugs- und Getriebeanalyse [AVL11b]. Es werden Lösungen für Planung und Entwicklung angeboten, die Parameteroptimierung und Komponentenverträglichkeit umfassen, um dabei den Anwender bei der Findung praktisch umsetzbarer Resultate zu unterstützen.

4.1.4 Excite

AVL Excite ist ein Tool, das sich gut eignet, um Akustik und Haltbarkeit eines Motors bzw. Antriebswelle zu berechnen [AVL11a]. Excite stellt eine breite Palette an Softwarelösungen zur Verfügung, die für die gesamte Entwicklungsphase eines Motors von Nutzen sind. Um dies optimal gewährleisten zu können, ist Excite in mehrere Module untergliedert:

- Bearing Analysis,
- Torsional Vibrations und
- Fatigue Strength Calculation.

4.2 Technischer Aufbau der Software

Das Softwareprodukt, das anhand dieses Projekts untersucht werden sollte, besteht aus mehreren Paketen (siehe 4.1). Jedes dieser Pakete besitzt mehrere Softwarelösungen die artverwandt sind (z.B. Abgassimulationen, Chasisberechnungen). Diese Lösungen werden zusätzlich in drei Teile aufgespalten und können als einzelne Features gesehen werden:

- Preprozessor,
- Postprozessor und
- Solver.

Diese Teile unterscheiden sich in ihrer Aufgabenverteilung. Preprozessoren beschreiben im Allgemeinen GUI- Komponenten, während Postprozessoren Optimierungs- Komponenten und die Solver die Lösung selbst (d.h. diese Komponente führt die Rechenoperationen durch) darstellen. Diese Komponenten sind getrennt voneinander ausführbar. Das heißt, dass zum Beispiel ein Mitarbeiter mit der GUI modelliert, während ein anderer mittels des Solvers Berechnungen anstellt. In den erhaltenen Logfiles werden diese Komponenten auch jeweils für sich ausgecheckt. Sie tauchen also auch als separates Feature in den Logfiles auf. Im folgenden Abschnitt wird noch genauer auf die Analyse der Logfiles eingegangen.

4.3 Implikationen und Aufbau der bereitgestellten Logfiles

Die von der Firma AVL bereit gestellten Logfiles beinhalten die Serverlogs des von der Firma intern genutzten Lizenzservers. Die AVL benutzt ihre hauseigene Software für Berechnungen, die sie auch weiter verkauft. Aus diesem Grund sind die Mitarbeiter der Firma nicht in ihrer Nutzung durch mangelnde Lizenzen beschränkt und haben freien Zugang auf die gesamte Produktpalette. Die Logfiles selbst sind Textdateien. Es werden Zeile für Zeile Checkout- und Checkin-Ereignisse eingetragen. Diese Ereignisse bezeichnen das Vergeben und Zurückgeben von Lizenzen. Abbildung 4.1 zeigt zwei Einträge aus dem Logfile. Man sieht wie der Aufbau der Einträge unterteilt wurde in Timestamp, Status, Feature und User.

Timestamp	Status	Feature	User
7:22:03	(avl) OUT:	"boost_gui"	cottetr@frpalwd508298
7:22:03	(avl) IN:	"boost_gui"	cottetr@frpalwd508298

Abbildung 4.1: Darstellung von zwei Logfileeinträgen unterteilt in Felder

4.4 Analyse der Logfiles

Die Analyse der Logfiles wurde anhand der zur Verfügung gestellten Daten durchgeführt. Dies sind, wie in Abschnitt 4.3 beschrieben, die Logfiles, die Preisliste sowie die Paketliste. Als Ziele für die Analyse der Logfiles wurden folgende Resultate angestrebt:

- Identifikation von Benutzerklassen,
- Identifikation von Featurepools,
- Identifikation von Auscheckverhalten und
- Erstellen von von Bewertungsgrößen und Benchmarks.

Bei Erreichen dieser Ziele soll es im Simulationsschritt möglich sein, gezielt das Auscheckverhalten einzelner Benutzerklassen in einem Featurepool für bestimme Tokensetups zu simulieren. Anhand der erstellten Benchmarks sollen die in weiterer Folge simulierten Tokensetups verglichen werden und auch die Möglichkeit bieten die Tokensetups untereinander zu bewerten.

4.4.1 Identifikation von Benutzerklassen

Aus den vorhandenen Logfiles wurden mehrere Benutzer ausgefiltert, die ein regelmäßiges Benutzerverhalten über den Beobachtungszeitraum (ca. drei Monate pro Logfile) aufwiesen. Dies geschah vorrangig um die Benutzer zu finden, die tatsächlich, auf täglicher Basis, mit dem jeweiligen Feature arbeiten. Im Anschluss wurden diese in Gruppen unterteilt:

- Hardcore User - Benutzer mit überdurchschnittlich hohem Auscheckverhalten,
- Average User - Durchschnittlich hohes Auscheckverhalten
- Casual User - Unterdurchschnittliches Auscheckverhalten.

Benutzer mit sporadischem Auscheckverhalten wurden nicht klassifiziert. Dies geschieht unter der Annahme, dass auch bei einem tatsächlichen Kunden diese Nutzer nur wenig Priorität haben.

4.4.2 Identifikation von Featurepools

Um die verschiedenen Tokensetups zu generieren, war es nötig zusammengehörige Features in einen Pool zu integrieren. Diese Pools sollten hauptsächlich dadurch charakterisiert sein, dass die dazugehörigen Features in einem logischen/brauchbaren Verhältnis zueinander stehen. Verwendet ein Benutzer ein Feature aus dem Featurepool, benötigt er mit hoher Wahrscheinlichkeit auch alle anderen Features.

4.4.3 Identifikation von Auscheckverhalten

Die Analyse des Auscheckverhaltens ist einer der wichtigsten Vorschritte zu den Simulationen. Nachdem nun Benutzer klassifiziert und die Featurepools erstellt wurden, gibt das Auscheckverhalten an, wie sich ein bestimmter Nutzer eines Features verhält. Das heißt, Dauer, Häufigkeit und Verteilung von ausgecheckten Features eines Benutzers über einen bestimmten Zeitraum gemessen. Diese Daten werden in weiterer Folge für die Simulation vereinfacht und schlussendlich angewandt.

4.4.4 Erstellen von Bewertungsgrößen und Benchmarks

Dieser Schritt der Analyse beinhaltet erstens die Findung von Bewertungsgrößen. Sie sind für Token Based Lizensierung sowie für die bisherig verwendeten Netzwerklizenzen als Vergleichsbasis anwendbar. Es wurden folgende Bewertungsgrößen identifiziert:

- Denial: Für jede Featureanfrage, die nicht erfüllt werden kann, weil keine Token bzw. Lizenzen frei sind, wird ein Denial gerechnet.
- Verzögerung: Jedes Feature wird abgehandelt, sobald ausreichend Token bzw. Lizenzen frei sind. Die Differenz zwischen Denial- und Abhandlungszeitpunkt ergibt die Verzögerung.
- Time used: Gesamtzeit, die ein Feature im Beobachtungszeitraum ausgecheckt ist.

Zweiter Punkt ist das Erstellen der Benchmarks. Da die vorhandenen Daten jedoch keinerlei Verzögerung oder Denial aufwiesen, weil die Benutzer in der AVL keiner Beschränkung der Benutzung ihrer Software unterliegen (siehe Abschnitt 4.3), mussten hier erst zusätzliche Maßnahmen ergriffen werden. Um realistische Werte für Denials und Verzögerung zu erhalten, wurde ein Dummyserver programmiert, der mittels der Logdaten

eine Beschränkung simuliert. Die daraus resultierenden Werte wurden für die Benchmarks herangezogen.

Dummyserver zur Berechnung von Denial und Delay

Die Daten der AVL, die in den Logfiles enthalten sind, stellen den normalen uneingeschränkten Lizenzgebrauch dar. Dieser uneingeschränkte Gebrauch ist für die Simulation von Kunden unrealistisch, da der Lizenzgebrauch im Normalfall durch die Anzahl der Lizenzen beschränkt ist. Nimmt man an, dass dieser uneingeschränkte Gebrauch den zeitlichen Gesamtbedarf darstellt und dieser sich nicht ändert, kann man durch setzen einer künstlichen Lizenzbeschränkung eine Vergleichsbasis schaffen und diese mit einem Tokensystem vergleichen. Der programmierte Dummyserver erfüllte diese Rolle. Es konnten verschiedene Lizenzgrenzen definiert werden und anhand der Logfiles simuliert werden. Jeder Logfileeintrag wird sequenziell ausgelesen. Checkouts erhöhen den Lizenzzähler während Checkins ihn verringern. Ist die maximale Anzahl an gleichzeitig laufenden Lizenzen erreicht, werden alle weiteren Checkouts abgelehnt und erst ausgecheckt nachdem ein Checkin erfolgt ist. Der Zeitraum zwischen den Denial und dem darauf folgenden Checkin wird ebenfalls gemessen und als Delay gewertet. Die Anzahl der Denials und der dazugehörige Delay können nun als Benchmark für Tokensysteme angewandt werden.

4.5 Vorgehensweise bei der Analyse

Wie in Abschnitt 4.3 beschrieben, sind die bereit gestellten Daten als Logfiles mit einem bestimmten Format übergeben worden. Aus diesem Grund mussten die Logfiles erst gefiltert werden, um die gewünschten Daten zu erhalten. Es wurden jene Zeilen isoliert, die jeweils den gewünschten Benutzer und Feature beinhalteten.

Nachdem die Benutzer und Features isoliert sind, werden die Zeiten der Checkouts und Checkins analysiert. Da durch das Logfile nicht ersichtlich ist, welches Checkin zu welchem Checkout gehört, wird hier auf einen einfachen FIFO Algorithmus (siehe Abbildung 4.2) zurückgegriffen. Dieser Algorithmus wurde gewählt, um eine neutrale Reihung der Features nach Checkoutzeit zu erhalten. Dies war insofern nötig, da zu diesem Zeitpunkt der Analyse noch keine Aussage über die Wichtigkeit einzelner Features getroffen werden konnte. Die Features sollten strikt nach ihren zeitlichen Bedarf analysiert werden. Hat man die Anzahl und die Dauer der Features ermittelt, kann man die Gesamtauscheckdauer des Features in Bezug auf den Beobachtungszeitraum ermitteln. Dies wird in weiterer Folge

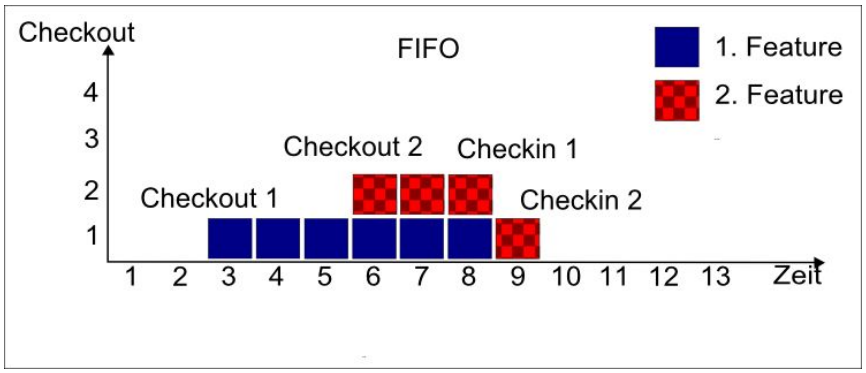

Abbildung 4.2: Darstellung des Parsevorgangs mittels FIFO Algorithmus

als Gesamtauslastung bzw. -sättigung definiert. Es wird auch festgestellt, wann bestimmte Features häufig benutzen werden und wie lange. Abbildung 4.3 zeigt, wie die Auscheckverteilungen für einzelne Benutzerklassen ausfallen können. Auf der X-Achse ist die Dauer des Checkouts aufgetragen. Sie gibt an wie lange eine Lizenz in Anspruch genommen wird. Auf der Y-Achse sieht man wie oft ein Feature für die entsprechende Dauer in Anspruch genommen worden ist.

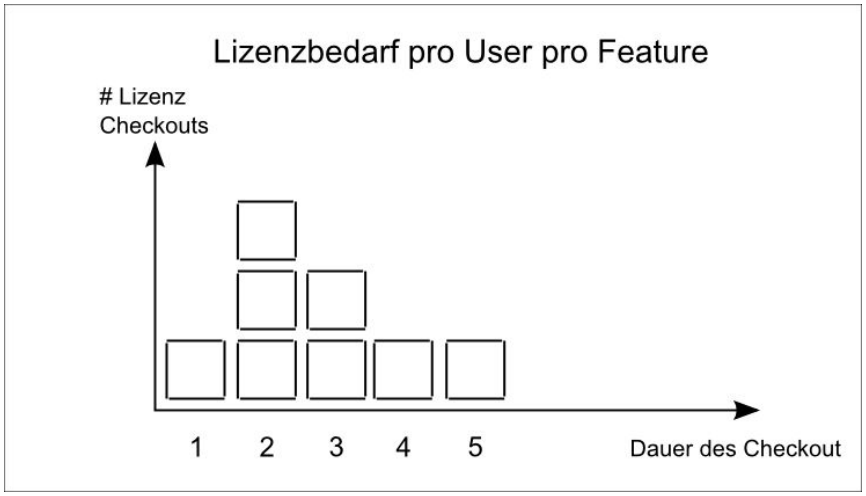

Abbildung 4.3: Lizenzbedarf eines Benutzers über den Beobachtungszeitraum

Zusätzlich geht aus den Logfiles noch hervor, wie oft ein Feature mehrfach zu einem bestimmten Zeitpunkt ausgecheckt ist. Die Wahrscheinlichkeit für einen Mehrfachcheckout ist vor allem wichtig für spätere Betrachtungen im Bezug auf Token Based Lizensierung. Abbildung 4.4 zeigt im Detail, wie ein Mehrfachcheckout verstanden wird. Wie man erkennen kann, sind zu Zeitpunkten 3, 4, 9 und 10 jeweils nur ein Feature der selben Art

ausgecheckt (dargestellt in blau). Zu Zeitpunkten 5 und 8 laufen jeweils zwei Features gleichzeitig und bei Zeitpunkten 6 und 7 jeweils drei.

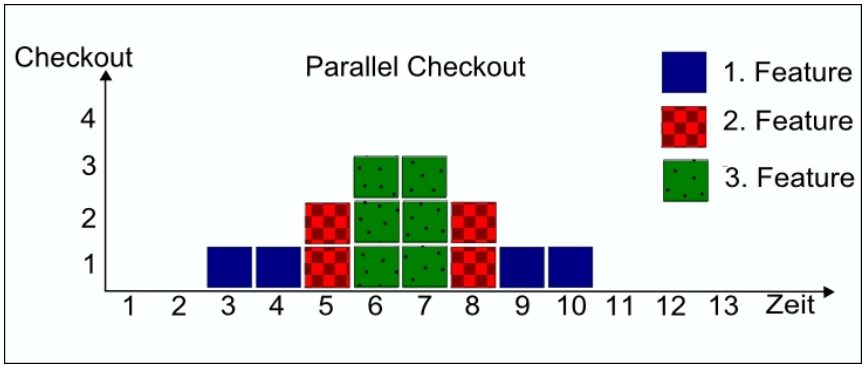

Abbildung 4.4: Darstellung des Parsevorgangs für gleichzeitig aktive Features

Aus diesen Parsevorgängen konnten folgende Werte für Features zugewiesen werden:

- Sättigung,
- Auscheckhäufigkeit,
- Auscheckdauer und
- Mehrfachcheckouts.

Anhand dieser Daten wurden nun Simulationen gefertigt, die für einen bestimmten Beobachtungszeitraum angewandt wurden.

4.6 Anwendungssimulationen

In diesem Abschnitt werden die Simulationen behandelt, die anhand der aus den Logfiles erhaltenen Daten durchgeführt wurden. Dabei wurden unterschiedliche Benutzer und Features miteinander kombiniert. Daraus kann man die Wahrscheinlichkeit errechnen, dass ein Feature zu einem bestimmten Zeitpunkt ausgecheckt wird und wie lange. Mit dieser Information ist es möglich, den Bedarf eines Features für einen Benutzer auf einer Zeitachse zu simulieren. Den Features auf den Zeitachsen wird nun ein Tokenwert zugewiesen. Abbildung 4.5 veranschaulicht diesen Vorgang.

Die Zuweisung des Tokenwerts erfolgt mittels Exhaustionsmethode (auch als Brute-Force-Methode bekannt), bei der sukzessive unterschiedliche Werte für die simulierten

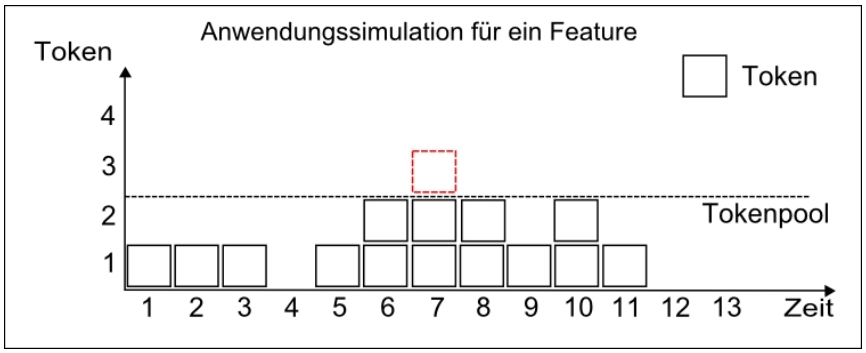

Abbildung 4.5: Darstellung der Simulation von Features mit Tokenwerten und Tokenpool

Features ausgewählt wurden. Hierbei wurde ein beliebiger Wertebereich als Begrenzung angenommen und eine Schrittlänge in der der Tokenwert variierte. Dies geschah, um einen Tokenbedarf zu jedem Zeitpunkt auf der Zeitachse ermitteln zu können, für eine Vielzahl an unterschiedlichen Tokensetups. Als Tokensetup wird eine Kombinationsmöglichkeit für die in der Simulation verwendeten Token genannt. Als Ergebnis dieser Anwendungssimulation wurde die Anzahl der Denials ermittelt, der Features, die durch einen zu geringen Tokenpool verursacht wurden. Ein Denial wird erzeugt, wenn ein Feature nicht zum sofortigen Zeitpunkt des Checkouts die nötigen Ressourcen zugewiesen bekommt, sondern erst zu einem späteren Zeitpunkt. Dies wird für alle Tokensetups durchgeführt, um diese im Anschluss miteinander vergleichen zu können. Doch alleine für sich, sind diese Daten noch nicht vergleichbar. Unterschiedliche Tokensetups haben verschiedene Tokenpools und Tokenwerte. Um eine Vergleichbarkeit zu schaffen, wird eine Monte Carlo Simulation für jedes getestete Tokensetup durchgeführt. Dies ermöglicht es einen %-Wert des Gesamtbedarfs eines Features für das jeweilige Setup zu ermitteln. Der Denial, der sich aus diesem %-Wert ergibt, kann dann verglichen werden. Der Vorgang, wie diese Monte Carlo Simulationen durchgeführt wurden, ist im folgenden Abschnitt beschrieben.

4.7 Monte Carlo Simulation zur Ermittlung des Tokenbedarfs

Im nächsten Schritt wird der Wert für den Tokenpool ermittelt. Es wird eine Monte Carlo Simulation durchgeführt, um zu ermitteln, wie oft eine bestimmte Anzahl an Token verwendet wird. Es werden anhand der ermittelten Wahrscheinlichkeiten, die aus den Logfiles ermittelt wurden, eine Reihe an Zufallsexperimenten durchgeführt. Diese Zufallsexperi-

mente sind Zeitpunkt bezogen. Das heißt, im Gegensatz zur Anwendungssimulation, die auch die Checkoutdauer in Betracht zieht, wird hier eine Stichprobe von einem Zeitpunkt genommen. Von Interesse sind die Token, die ein Feature zu diesem Zeitpunkt in Beschlag nimmt. Reiht man die Ergebnisse aus einer ausreichend hohen Anzahl an Zeitpunkten nach Tokenanzahl, kann man ermitteln, wie viel Bedarf an Token man mittels eines bestimmten Tokenpools abdecken kann [Jon10]. Wie man in Abbildung 4.6 sehen kann, besteht in dieser Simulation zu 10 Zeitpunkten ein Bedarf an Token. Der Tokenpool liegt bei zwei. Das heißt, es können 9 dieser 10 Fälle abgedeckt werden. Nur zu Zeitpunkt 7 reicht der Tokenpool nicht aus. In diesem Fall wäre der Tokenpool bei 90 %.

# Simulation	Bedarf	Tokenpool	Bedarf gedeckt
1	1	2	JA
2	1	2	JA
3	1	2	JA
4	0	2	JA
5	1	2	JA
6	2	2	JA
7	3	2	NEIN
8	2	2	JA
9	1	2	JA
10	2	2	JA
11	1	2	JA
12	0	2	JA
13	0	2	JA
14	0	2	JA
15	0	2	JA

Abbildung 4.6: Beispiel für eine Monte Carlo Simulation zur Berechnung des Tokenbedarfs für einen Tokenpool

4.8 Vergleich der simulierten Tokensetups

Um die einzelnen Simulationen miteinander vergleichen und bewerten zu können, wurde der Delay gemessen, der entsteht, wenn man einen Tokenpool darauf anwendet. Der Tokenpool wurde auf 70 % des gesamten Tokenbedarfs gesetzt. Es werden nun alle Features, die über dieser Grenze liegen, auf einen späteren Zeitpunkt verschoben. Die Zeit, die das Featurecheckout nach hinten verschoben wird, ist der Delay. Die Summe aus dem Delay für jedes Feature über dem Tokenpool ergibt den Gesamtdelay. Als zweite Vergleichsbasis

wurde der Verlauf des akkumulierten Tokengebrauchs herangezogen. Ein guter Verlauf entsteht, wenn zwischen einzelnen Tokenstufen keine großen Sprünge vorkommen. Ein stetiges Ansteigen wird bevorzugt, da damit auch der Mehrwert zusätzlicher Token stetig und nicht sprunghaft steigt.

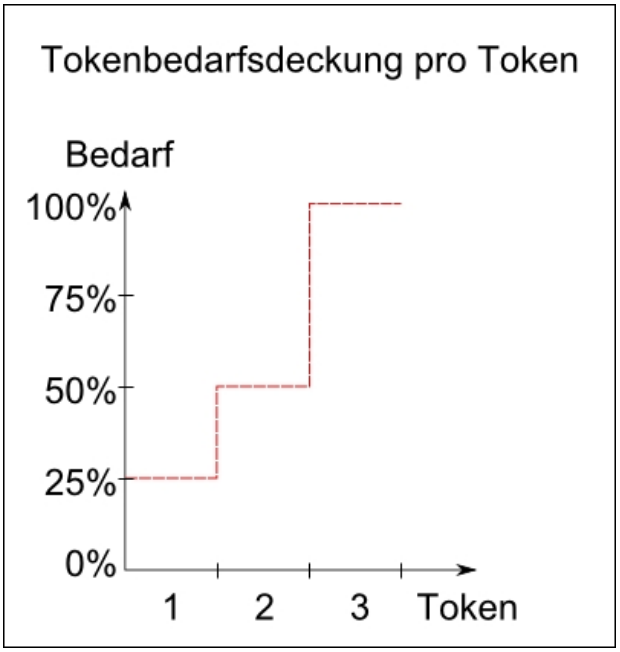

Abbildung 4.7: Beispielhafte Darstellung für akkumulierten Tokenbedarf

Wie man in Abbildung 4.7 beispielhaft sehen kann, wird mit einem Token 25 % des Bedarfs abgedeckt. Erhöht man auf zwei Token 50 %. Das heißt, zwischen null und zwei Token steigt die Bedarfsabdeckung proportional an. Bei drei Token jedoch, werden plötzlich 100 % des Bedarfs gedeckt. Der Sprung von zwei auf drei Token generiert den doppelten Mehrwert, wie ein Sprung von null auf ein oder von ein auf zwei. Ein Tokensetup, das so einen Bedarfsdeckungsverlauf aufweist, wäre schlecht. Gewisse Werte für den Tokenpool wären attraktiver für einen Kunden. In weiterer Folge würde er abgeschreckt werden, die nächst höhere Stufe für den Tokenpool zu erwerben, da sie verhältnismäßig weniger Mehrwert bringt.

4.9 Analyse der Ergebnisse aus den Anwendungssimulationen

In diesem Teil werden die einzelnen Ergebnisse der Tests analysiert. Dies umfasst die Lizenztests mit dem alten Jahreslizenzsystem sowie die Tests mit dem neuen Tokensystem. Hauptaugenmerk wird dabei auf den Delay gelegt, da diese für das Ergebnis am relevantesten sind. Bei Test 1-4 (siehe Abbildung 4.8) mit Jahreslizenzen werden in aufsteigender Wertigkeit Lizenzen vergeben. Die Verzögerung sinkt dabei stetig bis auf 0. Mit vier Lizenzen ist der Bedarf des Benutzers vollkommen gedeckt (Delay auf 0 d.h. max. vier parallele Chekouts im Paket). Dasselbe Vorgehen wird für weitere Benutzer angewandt. Hierbei stellt sich heraus, dass ein Benutzer mit 11 Lizenzen einen Power User darstellt, bzw. einer Weiterer mit geringem Bedarf von nur einem Lizenzpaket auskommt. Nun wurden die Benchmarks für den Vergleich mit den Tokensystemen gesetzt. Diese sind essenziell, um in weiterer Folge die Tokensysteme zu bewerten. Vor allem, wie sie im Vergleich mit dem alten System abschneiden. Nun werden die Token den einzelnen Features zugewiesen. Die hier vorgestellten Features sind pp2, Boost_main und Boost_gui und erhalten folgende Werte:

- pp2: 1200
- Boost_main: 6000
- Boost_gui: 2800

Bei Test 1-3 mit 10000 Token (siehe Abbildung 4.9) wird nun wieder die Verzögerung als Vergleichswert herangezogen. Mit der Annahme, dass man mit 10000 Token in etwa die Tokenmenge zur Verfügung hat, um das Lizenzpaket einmal auszuchecken, ist die Verzögerung sehr gering. Das lässt den Verdacht aufkeimen, dass das gewählte Tokensystem hier einen sehr dynamischen Effekt erzielt, der den Kunden stark bevorteilt. Test 2 und 3 setzen die Tokengrenze auf 8000 bzw. 6000 Token. Durch die geringere Anzahl der verfügbaren Token sollte nun die Verzögerung bei beiden Tests wesentlich höher ausfallen. Dies geschieht nur sehr marginal. Der gewünschte Effekt, dass sich die Features gegenseitig blockieren, tritt nicht bzw. nur in geringem Ausmaß ein. Dasselbe Phänomen tritt auch bei weiteren Tests mit unterschiedlichen Benutzern sowie auch Tokensetups auf. Auch bei Versuchen mit unterschiedlichsten Tokensetups werden die Ergebnisse nicht schlüssiger. Es scheint, dass jedes Tokensetup um ein Vielfaches bessere Ergebnisse im Bezug auf Delay erzielt, als die jeweils vergleichbaren Benchmarks (Netzwerklizenzen).

Ergebnisse Anwendungssimulation Netzwerklizenz			
Solution Pack	Betrachtungsdauer		
Boost Basic	45		
	Test 1	Test 2	Test 3
Verwendete Lizenzen pro Feature	1	2	3
boost_gui	7730396	7729960	0
pp2	3619088	0	0
boost_main	331949282	168699354	11053894

Abbildung 4.8: Auszug aus Test 1-3 der Anwendungssimulationen mit Netzwerklizensierung für Beispielbenutzer

Ergebnisse Anwendungssimulation Tokenlizenz			
Feature	boost_gui	pp2	boost_main
Token Kosten	10000	8000	6000
	Test 1	Test 2	Test 3
verwendeter Tokenpool	10000	10000	10000
boost_gui	177	177	57
pp2	0	0	0
boost_main	2700	2700	1001

Abbildung 4.9: Auszug aus Test 1-4 der Anwendungssimulationen mit Tokenlizensierung für Beispielbenutzer

Auf Grund der Ergebnisse dieser Tests ist die weitere Vorgehensweise in Frage gestellt. Es wurde angenommen, dass mittels Brute-Force-Methode ein geeignetes Tokensetup gefunden werden kann, um die Features bewerten zu können. Dies schien nicht der Fall zu sein. Es musste eine Erklärung für die Ergebnisse gefunden werden und eine neue Theorie zur Bewertung von Features.

4.10 Interpretation der Ergebnisse

In diesem Abschnitt wird näher auf die Ergebnisse der Testreihen aus dem vorangegangenen Abschnitt eingegangen. Es scheint, dass die Kombination von gewissen Features nur einen geringen Effekt auf den Gesamtdelay hat, wird ein Tokensystem angewandt. Der Delay, der als Benchmark für den Vergleich von Netzwerk- und Tokenlizensierung dienen sollte, trat für Tokenlizensierung vergleichsweise sehr gering auf. Eine Begründung dafür wurde aus den Logfiles entnommen (siehe Abbildung 4.10).

Wie man erkennen kann, wird dieses Feature sehr häufig ausgecheckt (im Sekundenbereich). Nun wird dies mit einem weiteren Feature verglichen (siehe Abbildung 4.11).

Auch hier gibt es sehr hochfrequente Checkout-/Checkin-Phasen. Zusätzlich gibt es noch längerfristige Checkouts (siehe Timestamp 6:33:19 - 14:19:30). Es stellt sich die Frage,

```
14:50:49 (avl) OUT: "boost_main" bodhades@ingurwd500042
14:51:00 (avl) OUT: "boost_main" bodhades@ingurwd500042
14:51:25 (avl) IN: "boost_main" bodhades@ingurwd500042
14:51:40 (avl) IN: "boost_main" bodhades@ingurwd500042
14:52:01 (avl) OUT: "boost_main" bodhades@ingurwd500042
14:52:53 (avl) IN: "boost_main" bodhades@ingurwd500042
14:53:14 (avl) OUT: "boost_main" bodhades@ingurwd500042
14:53:35 (avl) OUT: "boost_main" bodhades@ingurwd500042
14:54:09 (avl) IN: "boost_main" bodhades@ingurwd500042
14:54:15 (avl) IN: "boost_main" bodhades@ingurwd500042
```

Abbildung 4.10: Auszug aus den durch die Firma AVL zur Verfügung gestellten Logfiles für Feature boost_main

```
6:33:16 (avl) OUT: "boost_gui" bodhades@ingurwd500042
 6:33:17 (avl) IN: "boost_gui" bodhades@ingurwd500042
 6:33:19 (avl) OUT: "boost_gui" bodhades@ingurwd500042
14:19:30 (avl) IN: "boost_gui" bodhades@ingurwd500042
14:20:12 (avl) OUT: "boost_gui" bodhades@ingurwd500042
14:20:13 (avl) IN: "boost_gui" bodhades@ingurwd500042
14:20:14 (avl) OUT: "boost_gui" bodhades@ingurwd500042
14:35:19 (avl) IN: "boost_gui" bodhades@ingurwd500042
```

Abbildung 4.11: Auszug aus den durch die Firma AVL zur Verfügung gestellten Logfiles für Feature boost_gui

ob folgende Faktoren eine Auswirkung auf den Gesamtdelay haben:

- Die Checkoutdauer zweier Features ist sehr unterschiedlich.
- Die Anzahl der Checkouts ist sehr unterschiedlich.

Nimmt man an, dass diese Faktoren eine Auswirkung auf den Delay haben, muss eine Möglichkeit gefunden werden, diese angemessen zu bewerten. Diese Faktoren werden in weiterer Folge als Sättigung und Segmentierung bezeichnet. Bezieht man diese Werte mit ein, müsste die Interaktion zwischen zwei Features mit in die Bewertung von Token einbezogen werden. Als Beispiel könnte man folgendes Szenario betrachten. Ein Feature mit dem Wert 3000 wird eine Minute ausgecheckt. Ein zweites Feature mit dem Wert 1000 hat in derselben Minute 3 Checkouts zu je 10 Sekunden, die einander zeitlich nicht überlappen. In diesem Beispiel wird der Tokenpool nur mit 4000 Token belastet, hat aber eine Tokenausschöpfung von 6000. Nach dieser Theorie wären Features mit hoher Checkouthäufigkeit wesentlich effektiver in der Nutzung eines Tokenpools als die übrigen Features. Die Software der Firma AVL weist viele solcher Features auf. Abbildung 4.12 zeigt ein weiteres hochfrequentes Feature aus den Logfiles.

Die sehr hochfrequenten Features setzen in einem kurzen Zeitraum eine Menge Token

```
5:33:32 (avl) OUT: "pp2" bodhades@ingurwd500042
 7:05:36 (avl) IN: "pp2" bodhades@ingurwd500042
 9:46:30 (avl) OUT: "pp2" bodhades@ingurwd500042
 9:46:41 (avl) IN: "pp2" bodhades@ingurwd500042
11:25:39 (avl) OUT: "pp2" bodhades@ingurwd500042
13:58:23 (avl) IN: "pp2" bodhades@ingurwd500042
 6:00:44 (avl) OUT: "pp2" bodhades@ingurwd500042
 7:46:01 (avl) IN: "pp2" bodhades@ingurwd500042
 7:46:08 (avl) OUT: "pp2" bodhades@ingurwd500042
10:13:57 (avl) OUT: "pp2" bodhades@ingurwd500042
10:21:46 (avl) IN: "pp2" bodhades@ingurwd500042
11:21:29 (avl) IN: "pp2" bodhades@ingurwd500042
```

Abbildung 4.12: Auszug aus den durch die Firma AVL zur Verfügung gestellten Logfiles für Feature pp2

um. D.h. da ein solches Feature nur kurz Ressourcen blockiert, dieses blockieren aber die Wertgrundlage für ein Tokensystem darstellt, müssen diese Features entsprechend hoch bewertet werden. Kombiniert man diese Features mit langlebigen Features entstehen sehr starke Wertkontraste innerhalb eines Tokenpools (z.B.: Feature A kostet 10 Token, Feature B kostet 2000 Token). Dies wäre nicht sehr praktikabel, wird Feature A aus diesem Beispiel insgesamt nur ähnlich lang benutzt. Die Auscheckdauer (insgesamt) für A und B mag zwar gleich sein, aber die Häufigkeit der Checkouts verursachen einen großen Wertunterschied, der vor einem Kunden nur schwer zu erklären ist. Für die Firma AVL bedeutet das, das Features wie 'pp2' von ihren Werten her sehr billig sein müssten (lange CheckOuts). Dadurch haben sie tendenziell eine höhere Wahrscheinlichkeit verzögert zu werden (Delay) müssen sie zum gleichen Zeitpunkt bentutz werden wie ein langlebiges Feature.

4.11 Zusammenfassung und Fazit

In diesem Kapitel der Arbeit wurde versucht, Features anhand ihrer Nutzerdaten, die durch die AVL bereit gestellt wurden, zu bewerten. Es wurden Softwarepakete, die in einem Paket vereint sind, analysiert. Dies geschah, indem diese vom Rest der Features getrennt und nach Benutzern weiter unterteilt wurden. Die Benutzer wurden in Benutzergruppen unterteilt, je nach Intensität der Nutzung. Aufgrund der vorhandenen Daten konnte eine aussagekräftige Bewertung der Daten nicht erfolgen. Eine Bewertung der Features, die verkauft werden, ist anhand einer Analyse der Nutzungsdaten nur eingeschränkt möglich. Wie in [Mac03] beschrieben, ist es sehr schwierig eine Metrik zur Bewertung solcher Daten zu finden, da die Daten sehr stark variieren können. Des Weiteren ist es sehr schwierig einen Kompromiss zwischen Useranwendungen, die im Vordergrund sichtbar laufen, und Kernelfeatures zu

finden. Das subjektive Wertempfinden der Kunden kann hier sehr stark vom tatsächlichen Wert abweichen, da diese nicht bewusst benutzt werden.

4.11.1 Mögliche Maßnahmen zur Lösung des Bewertungsproblems von Features

Das Problem der Bewertung der einzelnen, sehr unterschiedlichen Features kann mit einem Tokenpoolsystem nicht direkt gelöst werden. Will man so ein System dennoch umsetzen, sollte die Architektur und die Anzahl der angebotenen Features möglichst einfach sein. Im Fall der AVL wären technische Änderungen an der Funktionsweise der Features anzuraten. Diese könnten wie folgt aussehen:

- Kopplung von Features: Die Tests haben gezeigt, dass die Kernelfeatures meist hochfrequent sind. Koppelt man nun diese Features an die GUI Features, könnte ein 'Beruhigungseffekt' auftreten. Features, die in der Regel länger ausgecheckt werden, werden also an kurzlebige gekoppelt. Dies soll eine Reduzierung der Komplexität herbeiführen und auch für mehr Stabilität sorgen.

- Blockieren der Features: Benutzte Token werden für einen Minimalzeitraum ausgecheckt. Auch wenn ein Feature nur 5 Sekunden benutzt wird, werden die Token erst nach 1 Minute wieder zurückgegeben. Die Schwäche dieses Konzepts wäre die Erklärbarkeit gegenüber dem Kunden, da in diesem Fall Lizenzanfragen geblockt werden, obwohl Lizenzen vorhanden wären. Als klaren Vorteil ist die Steuerbarkeit, den ein solches System dem Anbieter ermöglicht. Die Zufälligkeit durch den menschlichen Benutzer in Dauer und Häufigkeit der Checkouts kann mit einem solchen System von Anbieterseite her reduziert werden.

- Maßgeschneidertes System: Ein Abrechnungssystem, das die technischen Gegebenheiten der Software der Firma AVL mit einbezieht bzw. sogar zu seinem Vorteil nutzt.

Eine Mischung aus allen drei Möglichkeiten wäre erstrebenswert. Eine Umstrukturierung der bestehenden Features, um mehr Dynamik zu gewährleisten, sollte im Voraus gründlich evaluiert werden. Dabei sollten nicht nur technische Aspekte herangezogen werden. Die Entwickler können zwar eine gute Basis für die Möglichkeiten schaffen, es sollten aber ebenso marktwirtschaftliche Aspekte in die Preisfindung mit einfließen.

Kapitel 5

Konzept zur Erstellung von Tokensystemen mittels simulierter Daten

In diesem Kapitel wird erläutert, wie, nach dem erstem Versuch zur Erstellung eines Tokensystems anhand der Analyse mit Benutzerdaten (siehe 4.11 und 4.11), die Arbeit am Projekt neu konzipiert wurde. Hauptaugenmerk wurde dabei auf die Bewertung von Features gelegt. Es sollte eine Möglichkeit gefunden werden, Features anhand ihrer Eigenschaften, die relevant für eine Tokensystem sind, zu bewerten. Abbildung 5.1 zeigt Bewertungsgrößen, die für das Erstellen und Bewerten eines Tokensystems eine Rolle spielen können.

Die Abbildung zeigt die folgenden Aspekte:

- technische Eigenschaften der Features. Es wird nach allen technischen Einflussgrößen gesucht, die auf ein Tokensystem Einfluss haben könnten.

- Know How des Anbieters. Das Know How ist eine subjektive Bewertungsgröße für den Wert, den ein Feature bietet. Wie viel Wissen bzw. Forschung ist in seine Entstehung eingeflossen? Wie viele Mitbewerber mit gleichwertigen Produkten gibt es am Markt? Dies sind wichtige Faktoren.

- ökonomische Gegebenheiten am Markt. Hier fließen Faktoren wie Entwicklungszeit, Instandhaltungskosten (für z.B. Serviceapplikationen) und andere Nebenkosten, die gedeckt werden müssen, mit ein.

- andere Aspekte. Gegebenheiten, die nicht unter die bereits genannten Einflussgrößen fallen.

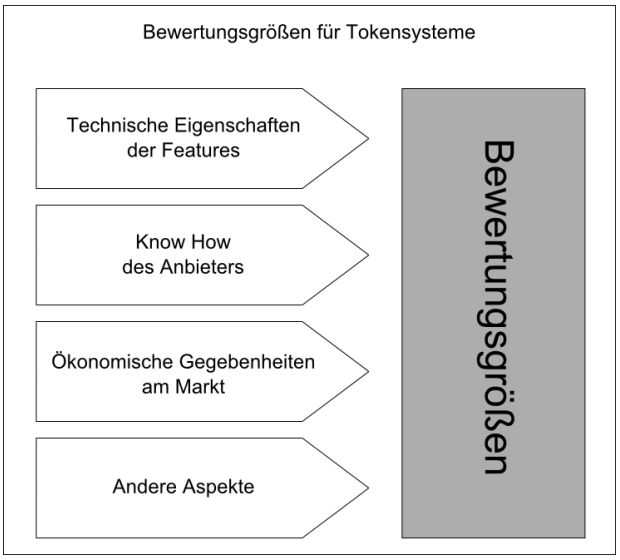

Abbildung 5.1: Einflussgrößen von Features zur Ermittlung von Bewertungsgrößen

In weiterer Folge wird nur nach den technischen Gegebenheiten von Tokensystemen bewertet. Wichtig ist, dass sie objektiv bewertbar und in Relation setzbar sind. Die technischen Gegebenheiten bestehen aus einem festgelegten Input und sollen damit einen Output in Form einer konkreten Bewertungsgröße liefern. In Abbildung 5.1 ist ersichtlich, dass technische Eigenschaften der Features einen Input darstellen und in dieser Arbeit als Features und deren Interaktionen definiert sind. Sie werden in Abschnitt 5.1.1 und 5.1.2 behandelt. Als das in der Abbildung als Bewertungsgröße gekennzeichnete Resultat ist der Delay gewählt worden. Die Beweggründe dahinter werden im folgenden Abschnitt näher erläutert.

5.1 Theoretischer Ansatz zur Bewertung von Features und deren Interaktion

Als theoretische Grundlage für die Erstellung eines Tokensystems galt, dass sich unterschiedliche Features eine Ressource teilen, um ausgecheckt werden zu können. Unter dieser Prämisse müssen nun die Einflussgrößen ausgesondert werden, die einen Einfluss darauf haben, wie stark die Ressourcennutzung ist. Hat man diese Einflussgrößen und deren Auswirkung auf die Ressourcennutzung erfasst, können diese als Input für einen Tokenberech-

nungsalgorithmus bzw. Featurebewertungsalgorithmus verwendet werden. Als Bewertungsgröße für die Ressourcennutzung wird der Delay herangezogen. Der Delay ist die Differenz zwischen dem Zeitpunkt, an dem ein Feature ausgecheckt werden soll und dem Zeitpunkt, an dem es tatsächlich ausgecheckt wird. D.h. wird ein Feature geblockt, weil zu wenig Ressourcen vorhanden sind, wird die Zeitspanne gemessen, bis diese frei werden.

5.1.1 Einflussgrößen für Features

Die elementarste Einheit in einem Tokensystem ist ein Feature. Um verschiedene Tokensetups untereinander vergleichen zu können, muss jeder Faktor in Betracht gezogen werden, der einen Einfluss auf die Features ausübt. Abbildung 5.2 zeigt welche Einflussgrößen im Laufe dieser Arbeit identifiziert wurden:

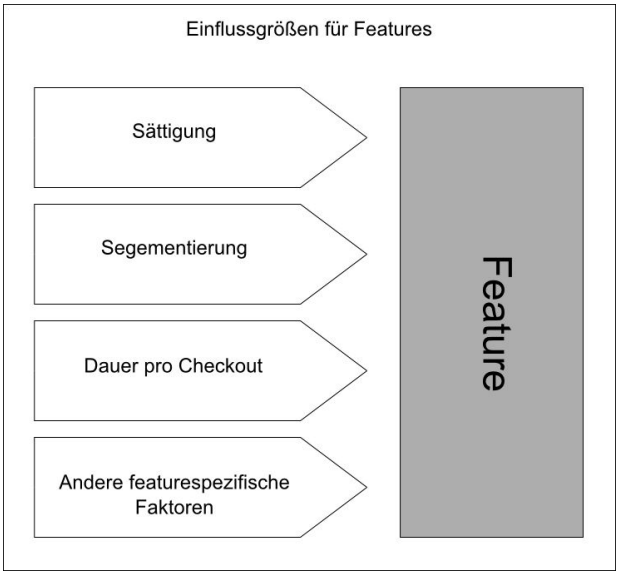

Abbildung 5.2: Einflussgrößen für Features

- Sättigung: Beschreibt die Gesamtdauer aller Checkouts eines Features. Die Sättigung wird als Prozentwert in Relation zum Beobachtungszeitraum dargestellt. Dies gewährleistet die Vergleichbarkeit über mehrere Testläufe hinweg. Als Prämisse für die Testläufe in Kapitel 6, die es zu untermauern gilt, wird angenommen, dass mit sinkender Sättigung auch der Delay sinkt.

- Segmentierung: Als Segmentierung eines Features wird die Anzahl der Checkouts in einem bestimmten Zeitraum bezeichnet. Da zwei Features theoretisch dieselbe

Sättigung haben können, aber unterschiedliche Segmentierungen, ist hier die Frage offen, wie sich das auf den Ressourcenverbrauch auswirkt. Wie in Kapitel 4.11 bereits erwähnt wird, muss gezeigt werden, ob und welchen Effekt die Segmentierung auf den Delay hat.

- Dauer pro Checkout: Diese Größe gibt an, wie lange nach einem Checkout die Ressourcen belegt sind, bis sie wieder frei werden.

- andere featurespezifische Faktoren: Diese Faktoren sind ein Oberbegriff für Einflussgrößen, die zwar einen Einfluss auf ein Feature haben können, aber für weniger wichtig erachtet wurden. Hier kann es sich um technische Einzelheiten handeln (z.B. Kernel- oder GUI Anwendungen) oder subjektive Faktoren, die nur schwierig bewertbar sind (z.B. logische (nicht technische) Abhängigkeiten zwischen Features).

Diese Einflussfaktoren wurden ermittelt, um für die folgenden Simulationen die nötigen Stellschrauben zu entwickeln, die zu einer erfolgreichen Tokenbewertung von realen Features führt.

Die Bewertung von Features ist jedoch nicht ausreichend, um effektiv Aussagen über ein Tokensystem treffen zu können. Bei einer reinen Featurebetrachtung, wird der Aspekt der Interaktion zweier Features komplett außer Acht gelassen.

5.1.2 Einflussgrößen für die Interaktion zwischen Features

In dieser Arbeit wird davon ausgegangen, dass der Delay, der von einem Tokensystem verursacht wird, nicht allein von den Eigenschaften seiner Features bestimmt wird. Dies lässt den Schluss zu, dass die Interaktion zweier Features ebenso einen Effekt auf den Ressourcenverbrauch ausübt. Die Frage, die sich stellt ist: Welche sind die Einflussgrößen, die die Interaktion zweier Features beeinflussen?

Wie Abbildung 5.3 veranschaulicht, wurden folgende Einflussfaktoren für die Interaktion zweier Features identifiziert:

- Anzahl der Features. Dieser Faktor gibt an, wie viele Features im Tokensystem interagieren. Mit jedem zusätzlichen Feature kommen auch dessen featurespezifische Einflussgrößen (5.1.1) zu tragen. Diese sind wichtig für die Berechnung der folgenden Einflussgrößen für Interaktion.

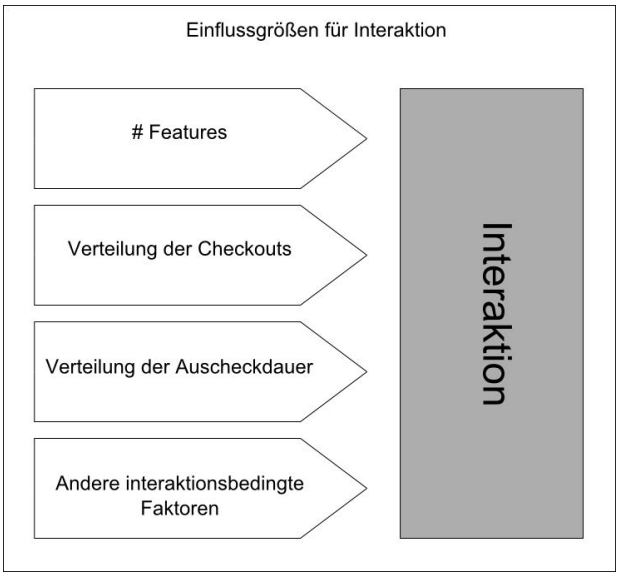

Abbildung 5.3: Einflussgrößen für Interaktion zweier Features

- Verteilung der Checkouts. Dieser Einflussfaktor soll den Gebrauch von Features über einen gewissen Beobachtungszeitraum hinweg simulieren. Die Annahme besteht darin, dass Features nicht immer gleichmäßig über den Tag hinweg benutzt werden. Es werden im Laufe dieses Abschnitts unterschiedliche Möglichkeiten präsentiert, wie diese Verteilungen aussehen könnten.

- Verteilung der Auscheckdauer. Auch für diese Einflussgröße gilt die Annahme, dass die Auscheckdauer für ein Feature nicht konstant ist. Es wird angenommen, dass die Features von Menschen angewandt werden, und nicht maschinell ausgecheckt werden, und daher Varianzen in der Auscheckdauer auftreten (dies ist besonders für Benutzeroberflächen relevant). Trotz dieser Varianz wird versucht ein Nutzungschema in der Auscheckverteilung zu finden, dass in weiterer Folge zur Berechnung von Token herangezogen werden kann.

- andere interaktionsbedingte Faktoren. Hier handelt es sich, wie bei den anderen featurespezifischen Faktoren, um alle Faktoren, die aus technischen oder subjektiven Gründen keine Rolle in den folgenden Experimenten spielen.

In den folgenden Abschnitten 5.1.2 und 5.1.2 wird auf die Verteilung der Checkouts und die Auscheckdauer näher eingegangen.

Verteilung der Checkouts

Die unterschiedliche Verteilung der Checkouts beruht auf der Annahme, dass Features nicht immer gleich oft benötigt werden. Dies umfasst, zum Beispiel, dass GUIanwendungen von Menschen benutzt werden. Daraus kann man schließen, dass solche Features sehr häufig tagsüber ausgecheckt werden und nur selten in der Nacht. Der umgekehrte Fall wäre, zum Beispiel, ein Feature das langwierige Berechnungen durchführt anhand einer einzelnen Eingabe. Solche Features werden häufig kurz vor Feierabend ausgecheckt, um Ressourcen nicht zu verschwenden. Die Darstellung einer solchen Verteilung erfolgt mittels Achsendiagramm. Die X-Achse gibt den Beobachtungszeitraum vor. Dieser kann je nach Zweck der Simulation beliebig lange ausfallen. Der Zweck sollte nach Möglichkeit der Abbildung eines realistischen Szenarios dienen. Dies kann, zum Beispiel, einen Arbeitstag, eine Arbeitswoche oder die Wochenendauslastung simulieren.

Die folgende Abbildung 5.4 zeigt, wie solche Checkoutverteilungen aussehen und welche Szenarien sie simulieren können. Für diese Darstellungen ist anzumerken, dass sie nur eine Annäherung an die vorgestellten Szenarien bieten sollen. Sie sind grob vereinfacht, um einen Überblick über bestimmte Verhaltensweisen zu erlangen. Eine mathematisch korrekte Darstellung des Auscheckverhaltens von Benutzern ist nur eingeschränkt möglich, da in einer solchen Simulation Ausnahmen wie Überstunden, Nachtarbeit, Schichtarbeit oder Krankenstand nicht berücksichtigt werden können. Ziel ist es lediglich zu zeigen, wie sich zwei Features mit einer bestimmten Verteilung im Beobachtungszeitraum verhalten.

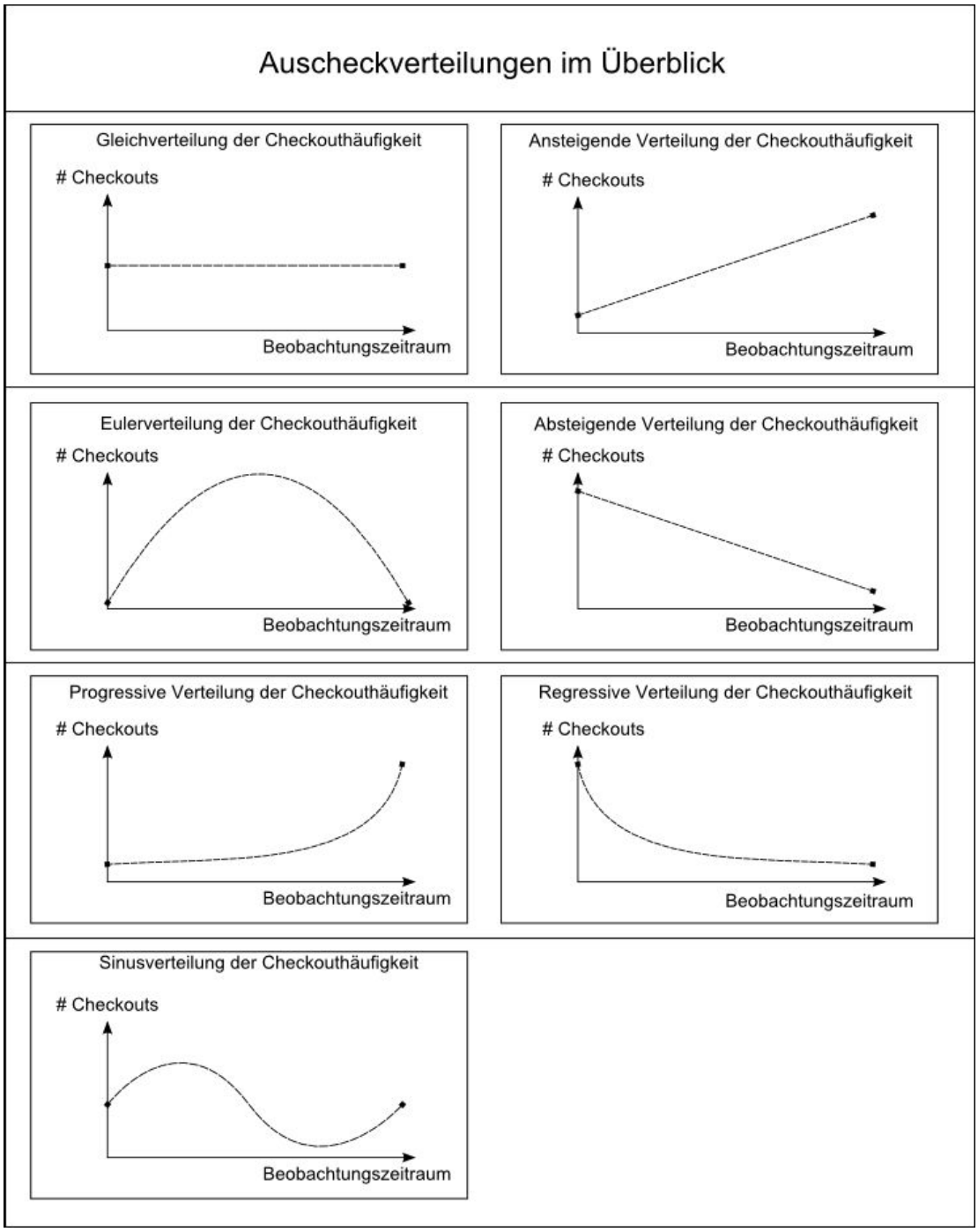

Abbildung 5.4: Überblick über mögliche Szenarien im Auscheckverhalten

Abbildung 5.4 zeigt folgende Möglichkeiten für Auscheckverhalten:

- Gleichverteilung,
- Ansteigende Verteilung,
- Absteigende Verteilung,
- Gaußverteilung,
- Progressive Verteilung,
- Regressive Verteilung und
- Sinusverteilung.

Die Gleichverteilung kann als Standardfall betrachtet werden. Die Checkouts werden nach Zufall gleichmäßig über den Beobachtungszeitraum hinweg verteilt. Ein Szenario, das geeignet wäre, eine solche Verteilung zu nutzen, wäre, zum Beispiel, ein relativ kurzer Beobachtungszeitraum, bei dem keine Schwankung in der Benutzung durch den Menschen angenommen wird. Als Beispiel kann man hier eine Firma mit Kernzeit annehmen. Die Kernzeit ist der Zeitraum an dem alle Mitarbeiter anwesend sind, und man wissen möchte wie viele Ressourcen dann benötigt werden. Die Gleichverteilung ist eher ungeeignet für die Darstellung von längerfristigem Ressourcenverbrauch, da in der Regel nicht gleichverteilt gearbeitet wird. Beispiele wie Tag-/Nacht Zyklus, Mittagspausen oder Kernzeiten werden nicht berücksichtigt.

Absteigende und ansteigende Verteilungen können für kurzfristige wie auch langfristige Simulationen Anwendung finden. Im kurzfristigen Bereich wären das, zum Beispiel, Schichtwechsel, Arbeitsende oder Anfang. Der Zweck darin liegt, den Übergang zwischen zwei unterschiedlichen Nutzungsphasen darzustellen. Im langfristigen Bereich könnte man eine solche Verteilung anwenden, um einen zyklischen Gebrauch der Features dar zu stellen. Weiß man, dass der Nutzungsgrad des Features über eine Abrechnungsperiode stark schwankt, kann man sich auf Grund dessen ein Bild vom Ressourcenbedarf machen.

Progressive und regressive Verteilungen dienen zur Darstellung von maschineller Featurenutzung (auch Batchvorgang genannt). Firmen verschieben aufwändige Rechenoperationen auf Nachtstunden, um Rechenkapazität und Lizenzen für die Tageszeit frei zu halten. Die Charakteristik liegt darin, dass innerhalb kurzer Zeit ein rapider Anstieg der Featureauslastung erfolgt.

Die Gauß- und Sinusverteilungen eignen sich gut für die Darstellung mittellanger Zeitspannen. Dies wäre, im Fall der Gaußverteilung, die Verteilung des Ressourcenbedarfs über einen Arbeitstag. Vormittags erscheinen die Mitarbeiter zur Arbeit und fangen an Features zu nutzen. Der Hauptbedarf ist zur Mittagszeit, wonach die Auslastung wieder sinkt. Die Sinusverteilung erweitert dieses Szenario noch um eine Nachtphase, zu der die Ressourcenauslastung nur gering ist (angenommen es gibt keine Batchvorgänge).

Anhand dieser Verteilungen ist es nun möglich ein gewisses Benutzerverhalten anzunähern. Es ist möglich, aus den einzelnen Verteilungen Stück für Stück eine Gesamtabbildung des Ressourcenverbrauchs zusammen zu fügen, die auch einem tatsächlichen Nutzer entsprechen könnte. Abbildung 5.5 zeigt beispielhaft, wie dies aussehen könnte.

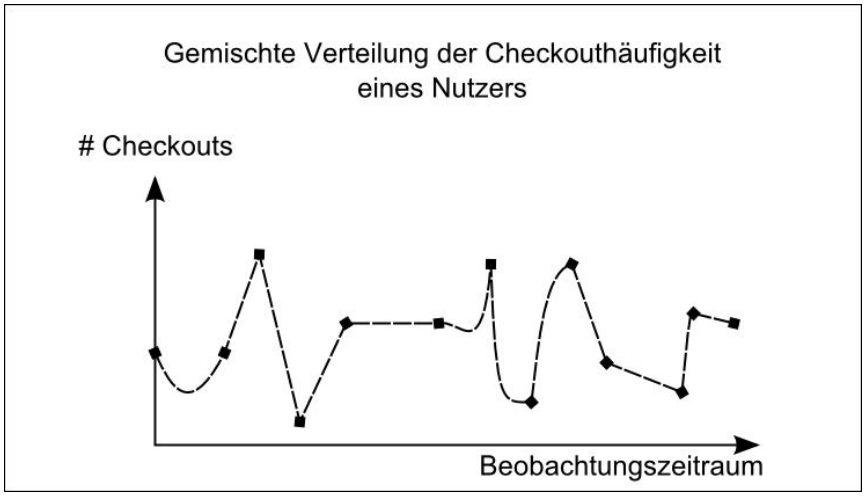

Abbildung 5.5: Mögliches zufällig auftretendes Nutzerverhalten für die Checkouthäufigkeit

Anhand dieser Verteilungen lässt sich nun das Benutzerverhalten modellieren. Man weiß, wie häufig Features in einem bestimmten Zeitraum ausgecheckt werden. Das Problem ist nun zu wissen, wie viele Ressourcen zu einem bestimmten Zeitpunkt im Beobachtungszeitraum verwendet werden. Das heißt, es reicht nicht einfach nur zu wissen, wann Ressourcen in Beschlag genommen werden, man muss auch den Zeitpunkt erfassen, wenn diese frei werden. Dies führt zur nächsten Einflussgröße der Interaktion zwischen Features: der Verteilung der Auscheckdauer.

Verteilung der Auscheckdauer

Die Einflussgröße Verteilung der Auscheckdauer ist im Gegensatz zur Verteilung der Checkouts nicht hauptsächlich durch die Benutzung durch den Menschen geprägt, sondern ori-

entiert sich eher an technischen Aspekten. In dieser Arbeit wird diese Verteilung ebenfalls als Achsendiagramm dargestellt. Die X-Achse gibt die Auscheckdauer an, während die Y-Achse die Wahrscheinlichkeit des Auftretens eines Checkouts mit der entsprechenden Dauer angibt. Abbildung 5.6 zeigt beispielhaft wie dies aussieht.

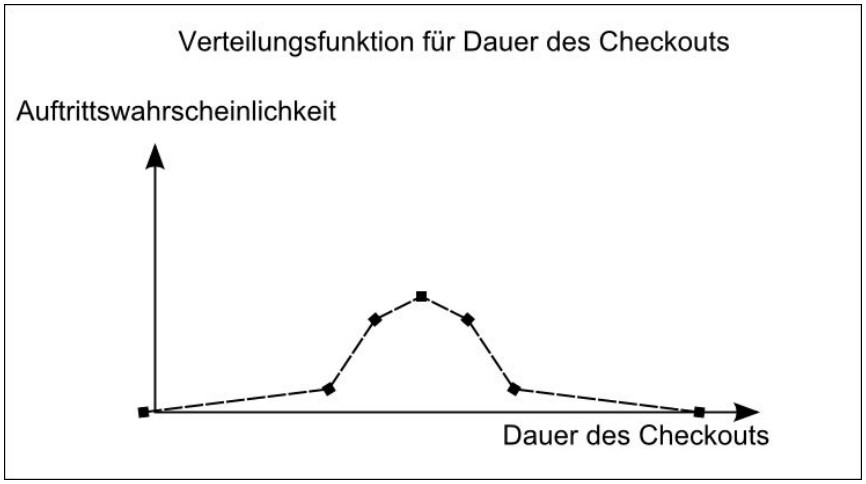

Abbildung 5.6: Beispielhafte Darstellung für eine Verteilung der Auscheckdauer

Im Unterschied zu den unterschiedlichen Verteilungsfunktionen die für die Darstellung der Checkouts herangezogen wurden, ist die Vielfalt hier eher eingeschränkt. Im Prinzip sollte jede Verteilung für Auscheckdauer sich ähnlich verhalten wie in Abbildung 5.6 dargestellt. Das heißt, die Verteilung ist glockenförmig (Gaußverteilung). Die einzigen Unterschiede sind die Position der Glocke auf der Skala und deren Wölbung. Die Position gibt an ob, ein Feature generell lang oder kurzlebig ist, während die Form der Glocke die Varianz in der Dauer angibt. Hier ist noch anzumerken, dass bei erhöhter Varianz für die Dauer auch der Ungenauigkeitsfaktor für die Bewertung ansteigt. Abbildung 5.7 zeigt wie eine optimale Verteilung der Auscheckdauer aussieht. Hier ist zu sehen, dass das betreffende Feature nur einen einzigen Wert für die Auscheckdauer bei Punkt X besitzt. Es gibt keine glockenförmige Auscheckverteilung und somit keine Varianz in der Dauer eines ausgecheckten Features. D.h. jedes ausgecheckte Feature dauert X Zeiteinheiten. Ein Beispiel für eine schlechte Auscheckverteilung ist in Abbildung 5.8 zu sehen.

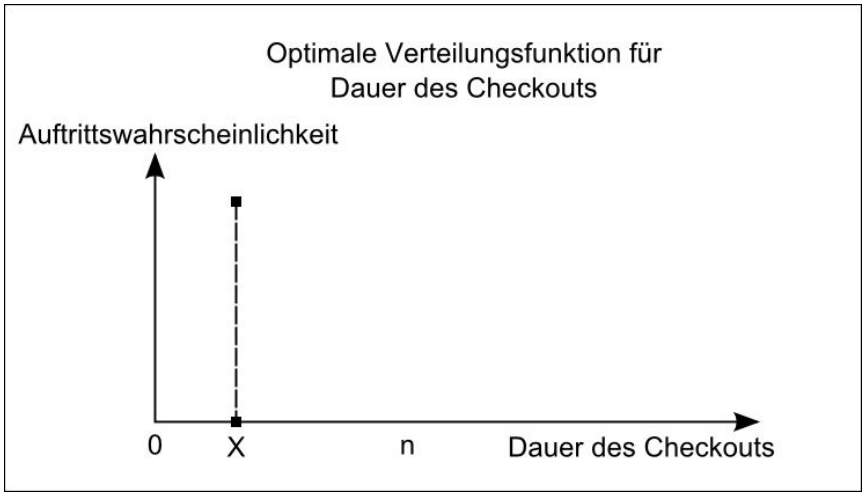

Abbildung 5.7: Beispiel für eine optimale Verteilung der Auscheckdauer

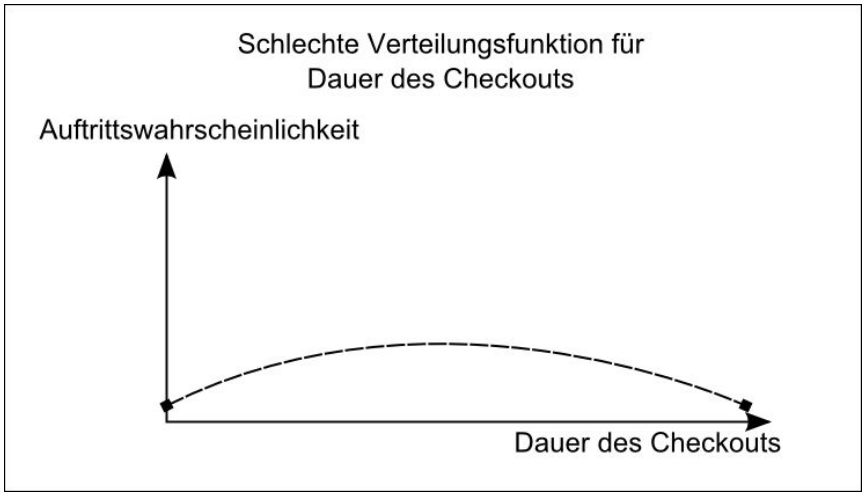

Abbildung 5.8: Beispielhafte Verteilung für eine stark variierende Checkoutdauer

Diese Verteilung wird in diesem Konzept als schlecht charakterisiert, weil sie einer einfachen Modellierung eines Tokensystems im Weg steht. Der Grund dafür ist, dass dieses Experiment den Zweck verfolgt, ein möglichst einfaches Modell für die Interaktion von Features zu erstellen. Hat man nun ein Feature, das eine sehr starke Varianz in der Auscheckdauer aufweist, ist es schwieriger darüber eine konkrete Aussage zu treffen. Dies gilt auch für andere Verteilungen, die keine definitive Aussage über die Resultate zulassen. Dies kann, zum Beispiel, bei Features sein, die unterschiedlich benutzt werden, und je nach Art, eine unterschiedliche Charakteristik für die Auscheckdauer aufweisen. Solche Fälle sollten bewusst ausgelassen werden.

5.2 Durchführung des Experiments

In diesem Abschnitt wird erläutert, wie der theoretische Ansatz umgesetzt wird, um nun ein konkretes Ergebnis zur Bewertung von Tokensystemen zu erhalten. Abbildung 5.9 zeigt schematisch den Ablauf eines Experiments.

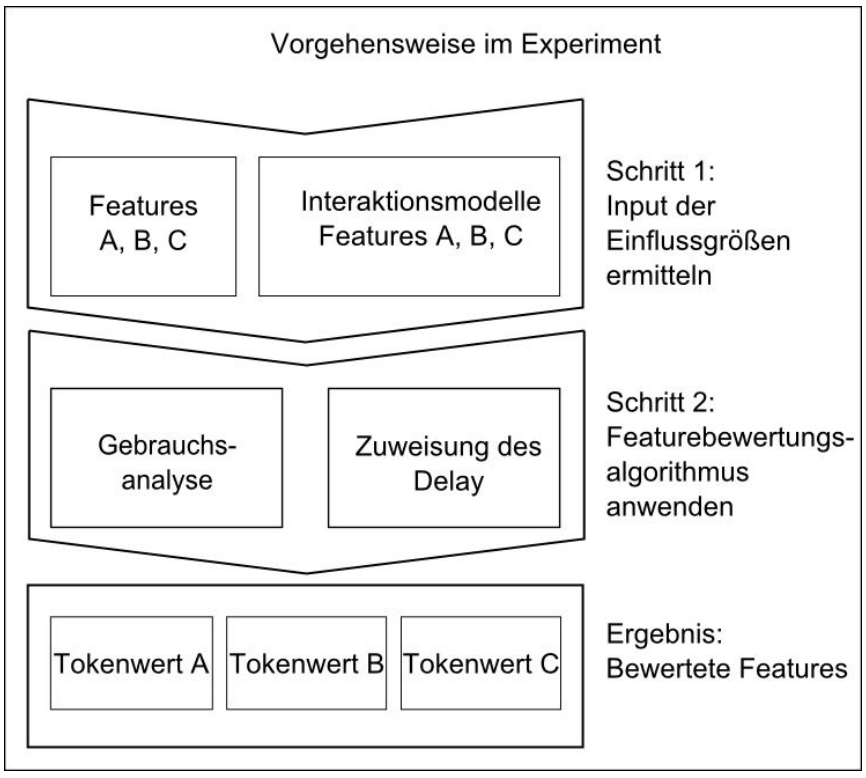

Abbildung 5.9: Ablauf eines Experiments zur Tokenbewertung eines Features

5.2.1 Schritt 1: Input der Einflussgrößen ermitteln

Im ersten Schritt wird der Input definiert. Das heißt, es werden alle Features, die für einen Tokenpool vorgesehen sind, wie in Abschnitt 5.1.1 beschrieben, bewertet. Es sind also Sättigung, Segmentierung und Dauer des Checkouts für die Inputfeatures bekannt. Im weiteren Verlauf wird nun jedem Feature ein Interaktionsmodell zugewiesen, das seinem Verhalten am ehesten entspricht. Ist der Input definiert, wird die nächste Phase im Experiment gestartet.

5.2.2 Schritt 2: Featurebewertungsalgorithmus anwenden

In diesem Abschnitt wird beschrieben, wie die Simulation der Inputdaten abläuft. Abbildung 5.10 zeigt das Zusammenspiel der ermittelten Einflussgrößen für Features und deren Interaktion, um eine Anwendungssimulation zu erzeugen. Man kann erkennen, dass in dieser Abbildung eine Gaußverteilung für Checkouts und ein konstanter Wert für die Auscheckdauer (in diesem Fall X = 1) gewählt wurde. Das Ergebnis dieser Anwendungssimulation ist eine Zeitachse über den Beobachtungszeitraum auf der X-Achse und die Anzahl des gleichzeitigen Gebrauchs des Features auf der Y-Achse. Die Anzahl der Checkouts und die Dauer wird von den featurespezifischen Eigenschaften vorgegeben (siehe Abschnitt 5.1.1).

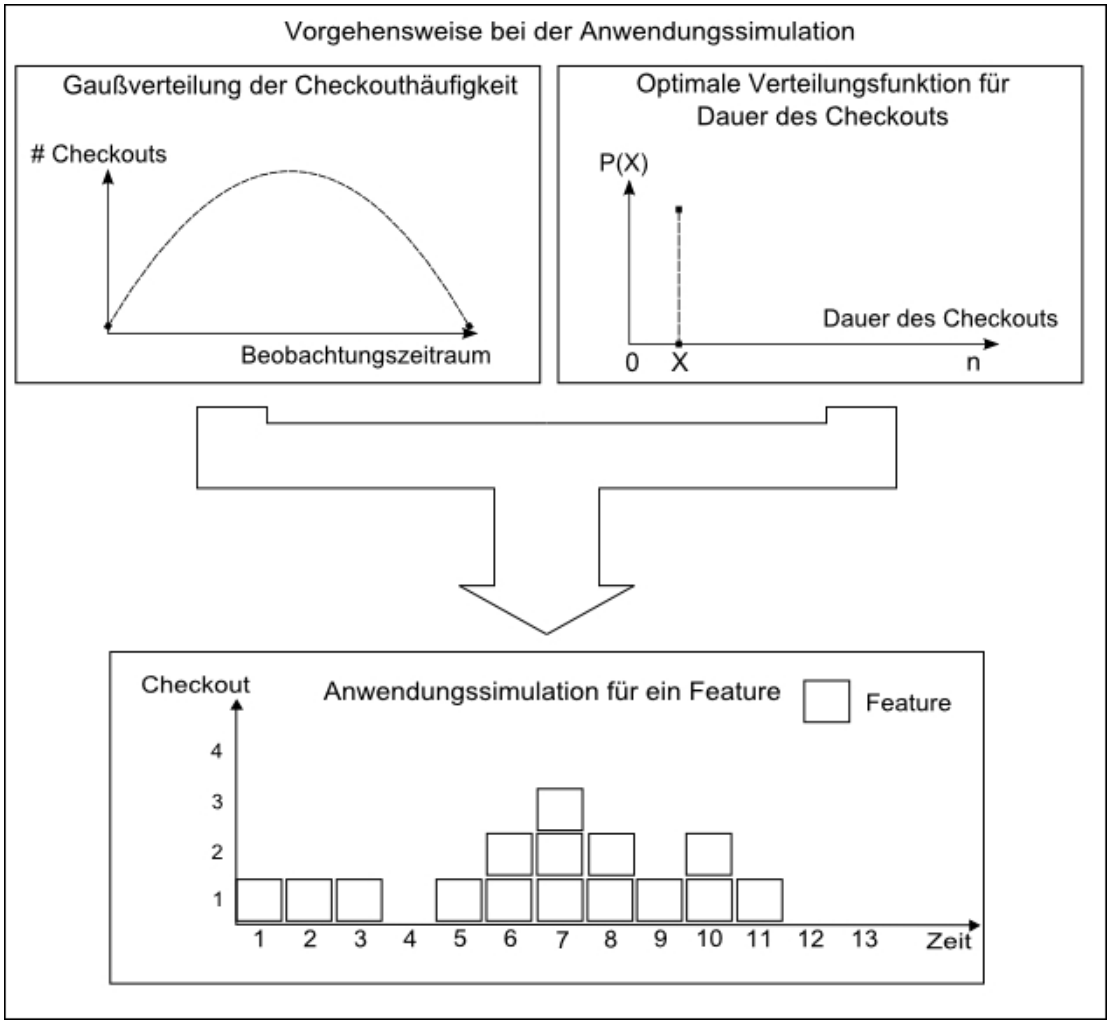

Abbildung 5.10: Vorgehensweise um aus einem Feature und dessen Interaktion eine Gebrauchssimulation zu erstellen

Man kann erkennen, dass die Verteilung der ausgecheckten Features in etwa der Gaußkurve entspricht, die von der Checkoutverteilung angewandt wurde. Sie stimmt jedoch nicht komplett überein. Dies ist auf das Benutzen von Wahrscheinlichkeiten zurückzuführen. Deswegen werden mehrere dieser Zufallsexperimente durchgeführt. Berechnet man nun für jedes einzelne den verursachten Delay und nimmt davon den Durchschnitt, erhält man einen Erwartungswert für den verursachten Delay mit dem gegebenen Input. Der Delay aus einem Experiment wird wie folgt errechnet: Nachdem die Features nach dem Zufallsprinzip auf der Zeitachse der Gebrauchssimulation aufgetragen wurden, wird nach Konflikten gesucht. Ein Konflikt entsteht, wenn zwei Instanzen eines Features gleichzeitig auf eine Ressource zugreifen wollen, also, gleichzeitig ausgecheckt werden müssen. Der Delay ist nun die Anzahl der Zeiteinheiten vom Zeitpunkt des Konflikts bis zu dem Zeitpunkt an dem wieder Ressourcen frei sind, um das Feature auszuchecken.

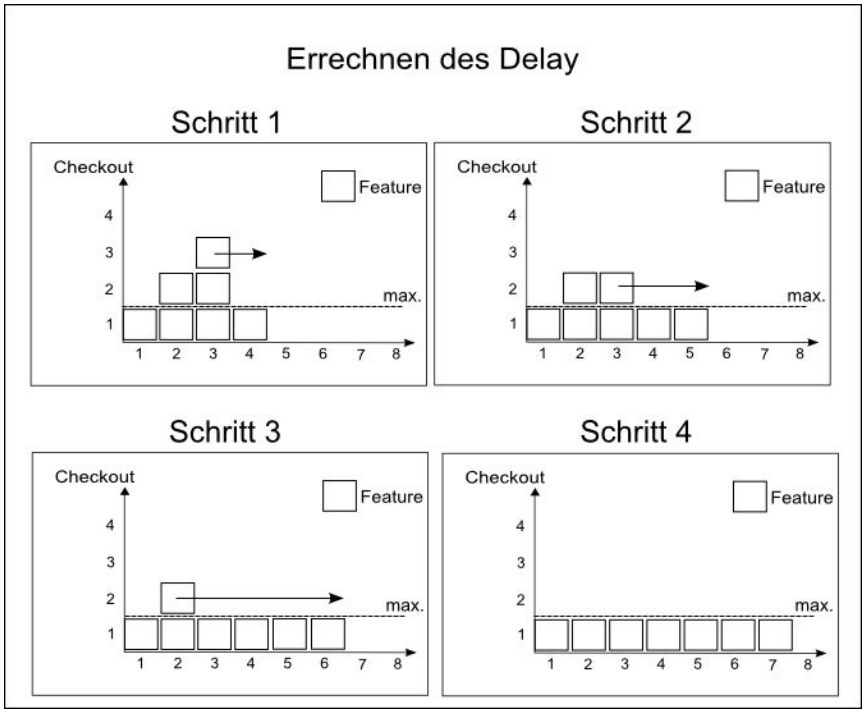

Abbildung 5.11: Vorgehensweise bei der Errechnung des Delay für eine Gebrauchssimulation

5.2.3 Schritt 3: Bewertete Features ermitteln

Wie in Abbildung 5.11 ersichtlich ist, wird von Schritt 1 bis Schritt 3 der Delay (dargestellt als Pfeil) gemessen. Die Summe aus allen Messungen ergibt den Gesamtdelay. Um die Vergleichbarkeit für unterschiedliche Experimente zu gewährleisten, wird der Gesamtdelay durch die Anzahl der Checkouts dividiert (Segmentierung). Der resultierende Wert ist der Delay pro Checkout und dient als Vergleichsbasis. Die Messung der Checkoutdauer ist nicht die einzige Aktion, die dabei durchgeführt wird. Es wird bei jeder einzelnen Messung das betreffende Feature auf die freie Position verschoben (siehe Abbildung 5.11).

Diese Gebrauchsanalyse wird nun für alle Features angewandt, die für einen Tokenpool vorgesehen sind. Die Gebrauchssimulationen über mehrere Features werden nun zusammengeführt und auf gleiche Weise wie in Abbildung 5.11 der Delay berechnet. Das Ergebnis aus diesem Verfahren ist eine Bewertung der verwendeten Features nach dem Delay, den sie in einem Tokensystem verursachen würden.

5.3 Zusammenfassung und Fazit

In diesem Kapitel wurde die Vorgehensweise beschrieben, die angewandt wurde, um das Verhalten von unterschiedlichen Features mit jeweils unterschiedlichen Eigenschaften in einem Tokensystem zu simulieren und zu bewerten. Es wurde dabei abgesteckt, wie die Einflussgrößen definiert werden, die einen Effekt auf die Bewertung haben. Hierbei wurde das Hauptaugenmerk auf den technischen Aspekt gelegt und Features und deren Interaktion ausgewählt. Für Features sind diese Einflussgrößen Sättigung, Segmentierung und Dauer des Checkout (siehe 5.1.2). Die Interaktion wird anhand der Verteilung der Checkouts und der Verteilung der Checkoutdauer (siehe Abschnitte 5.1.1 und 5.1.2) definiert. Sind diese Werte bekannt, wird in weiterer Folge ein Experiment durchgeführt, das den zu erwarteten Gebrauch eines Features simuliert. Dies wird anhand mehrfach durchgeführter Zufallsexperimente durchgeführt (siehe Abschnitt 5.2). Der aus diesen Zufallsexperimenten abgeleitete Delay wird als Bewertungsgrundlage für die Bewertung von technischen Aspekten von Features angewandt. Zusätzlich zur Bewertung der Features muss noch eine Tokenbedarfsanalyse gemacht werden. Dies geschieht auf die selbe Art und Weise wie in Abschnitt 4.7 mittels Monte Carlo Simulation. Es wird eine ausreichend hohe Anzahl an Zufallsexperimenten mit den ermittelten Tokenwerten durchgeführt. Die einzelnen Kombinationsmöglichkeiten der Features und deren summierte Tokenwerte ergeben einen bestimmten Bedarf. Nun kann der Tokenpool entsprechend des Bedarfs festgelegt werden.

Kapitel 6

Durchführung der Experimente mittels simulierter Daten

In diesem Kapitel wird der zweite Teil der durchgeführten Simulationen im Detail beschrieben. Die in Kapitel 5 vorgestellten Methoden wurden hier angewandt, um eine Bewertung von unterschiedlichen Features und deren Interaktion miteinander ermitteln zu können.

6.1 Theorie

Es hat sich in den vorangegangen Simulationen abgezeichnet, dass, bevor man Features in ein System wie ein Tokensetup einbauen kann, man sich auf grundlegender Ebene mit dem Verhalten der einzelnen Features auseinander setzen muss. Die Einflussgrößen und deren Auswirkungen auf ein Tokensystem sollen näher erforscht werden, und in weiterer Folge in einen anwendbaren Algorithmus umgesetzt werden. Im Detail sind folgende Fragen aufgetaucht:

- Wie verhält sich ein hoher Unterschied in der Segmentierung von Features?
- Wie verhält sich ein hoher Unterschied in der Sättigung von Features?
- Sind Segmentierung und Sättigung von einander abhängig?

Die zu Grunde legende Annahme in dieser Simulation ist, dass zwei Features mit zu unterschiedlichen Eigenschaften nicht in einem Tokenpool sein sollten. Sind zwei sehr unterschiedliche Features im gleichen Tokenpool, so tritt ein starker Effektivitätsverlust im

gesamten Tokensystems ein. Die untersuchten Eigenschaften sind in diesem Fall Segmentierung (Auscheckhäufigkeit) und Sättigung (Auscheckdauer). Grundsätzlich baut ein Tokensystem auf die Tatsache auf, dass sich die Features, die darin enthalten sind, in einer bewertbaren Relation zu einander befinden. Diese Relation stellt die Grundlage für die Vergabe der Tokenwerte dar. Diese Relationen werden aufgrund der Auscheckhäufigkeiten und Auscheckdauer der Features errechnet. Die Frage, die sich nun stellt ist: "Wie sehr können sich Features unterscheiden, um noch sinnvoll gemeinsam in ein Tokensystem aufgenommen werden zu können?" Die Sinnhaftigkeit ist zwar eine subjektive Größe, da sie von Fall zu Fall neu bewertet werden muss. Grundsätzlich lässt sich jedoch die Aussage Treffen, dass je Unterschiedlicher zwei Features in ihrer Bewertung sind, desto weniger Sinnvoll sind diese zusammen in einem Tokensystem.

6.2 Vorgehensweise

Wie in Kapitel 5 beschrieben, spielen in einem Tokensystem mehrere Einflussgrößen eine Rolle. Für dieses Experiment waren folgende featurespezifische und interaktionsbedingte Einflussgrößen relevant (siehe 5.1.1 und 5.1.2):

- Gesamtbeobachtungszeitraum,
- Anzahl der Features,
- Verteilung der Checkouts,
- Verteilung der Chekoutdauer,
- Sättigung und
- Segmentierung.

Um die featurespezifischen Einflussgrößen nun bewerten zu können, wurden mehrere Gebrauchssimulationen als Teststrecken durchgeführt. Pro Teststrecke wurde nur eine einzige Variable geändert und der Delay berechnet. Abbildung 6.1 zeigt, wie so eine Teststrecke aufgebaut ist. Wie man erkennen kann, werden nur zwei unterschiedliche Features zur Bewertung herangezogen: A und B. Diese beiden Features haben eine Gesamtsättigung (die Summe der Sättigung von Feature A und Feature B) und einen Wert für Segmentierung (in diesem Fall jeweils 1). Will man nun ermitteln, welchen Effekt die Segmentierung auf

den Delay hat, wird diese Variable nun für ein Feature verändert. Abbildung 6.2 zeigt eine Erhöhung der Segmentierung für Feature B auf 5 während die anderen Variablen gleich bleiben. Das heißt, die Gesamtsättigung für Feature A und B, die Segmentierung für A und der Gesamtbeobachtungszeitraum bleiben ident.

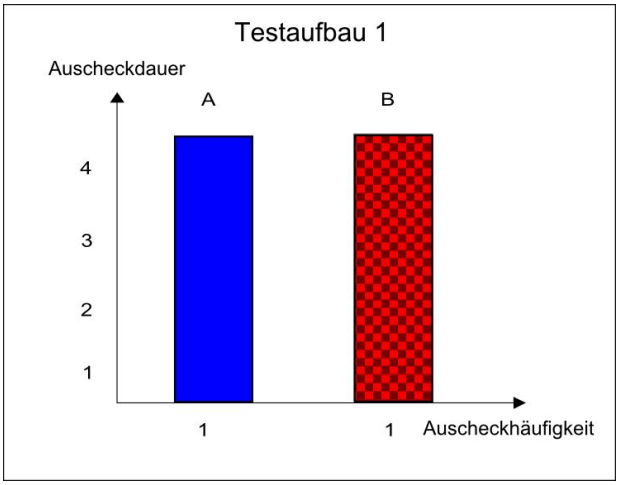

Abbildung 6.1: Testaufbau für zwei gleich segmentierte Features

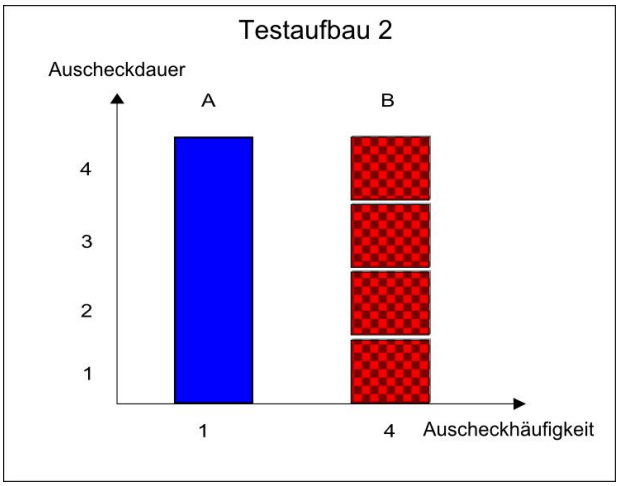

Abbildung 6.2: Testaufbau für zwei unterschiedlich segmentierte Features

Dieser Vorgang wird nun für diverse Kombinationen aus Sättigung und Segmentierung für zwei Features durchgeführt. Die Teststrecken reichen von einer 20 % Gesamtsättigung bis zu einer 100 % Gesamtsättigung. Dies bedeutet in weiterer Folge, für, zum Beispiel, eine Gesamtsättigung von 50 %, dass alle Kombinationen aus der Sättigung von A und B die 50 % ergeben, einen Testlauf darstellen. Dies geschah in 10 %-Schritten: z.B. 40 %

zu 10 % und 30 % zu 20 %. Ähnlich wurde bei der Segmentierung verfahren. Hier wurde mit Segmentierungsrelationen gearbeitet, die von einer 1:1 Relation bis hin zu einer 1:10 Relation von Feature A zu B untersucht wurden.

Um nun eine möglichst neutrale Bewertung von Segmentierung und Sättigung aufstellen zu können, wurden für die Einflussgrößen für Interaktion zwischen Features neutrale Verteilungen angestrebt. Diese sollten über den Verlauf der einzelnen Teststrecken, wenn möglich, keinen oder einen konsistenten Einfluss auf die featurespezifischen Einflussgrößen haben. Eine Gleichverteilung ist für diesen Zweck optimal. Eine zufällig verteilte Auscheckwahrscheinlichkeit über den Beobachtungszeitraum schien passend, um eine generelle Aussage über das Verhalten zweier Features treffen zu können. Die Verteilung der Auscheckdauer wurde ebenfalls so simpel wie möglich gehalten. Die Varianz wurde auf 0 gesetzt, das heißt, es gibt nur eine einzige Auscheckdauer für einen Testlauf. Diese Auscheckdauer wird bestimmt, indem die Sättigung des Features durch seine Segmentierung dividiert wird. Abbildung 6.3 veranschaulicht dies, wenn man für X = Sättigung/Segmentierung annimmt.

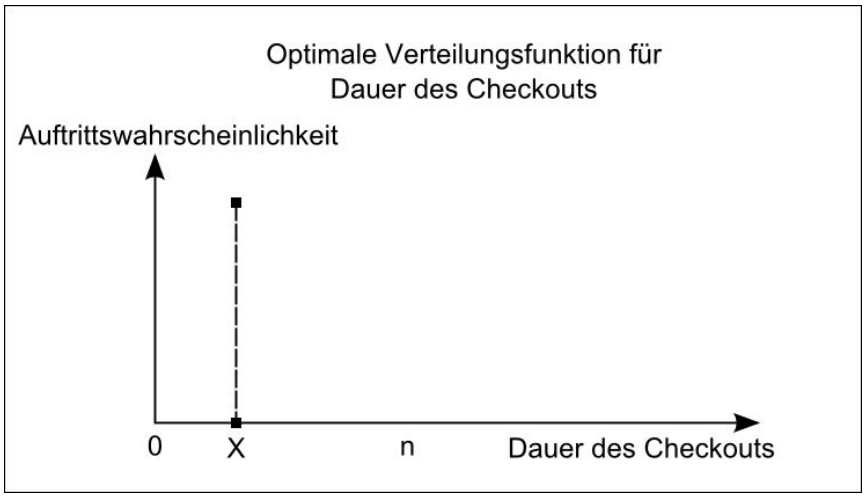

Abbildung 6.3: Beispiel für eine optimale Auscheckverteilung

6.3 Durchführung der Testläufe

Nachdem die Inputvariablen für die Testläufe festgelegt sind, wird, wie in Kapitel 4.6 beschrieben, eine Gebrauchssimulation für Feature A und Feature B durchgeführt. Abbildung 6.4 und 6.5 veranschaulichen dies beispielhaft.

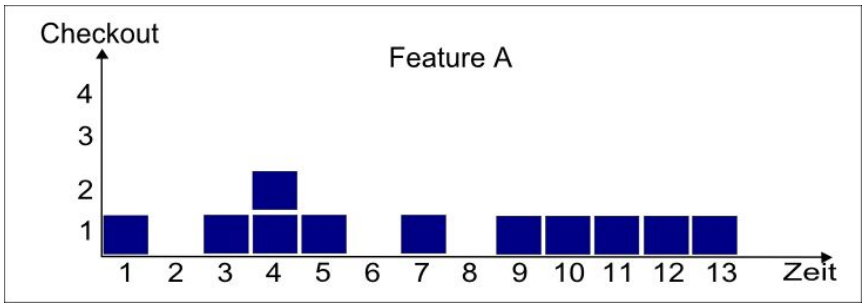

Abbildung 6.4: Beispiel für mögliches Auscheckverhalten von Feature A

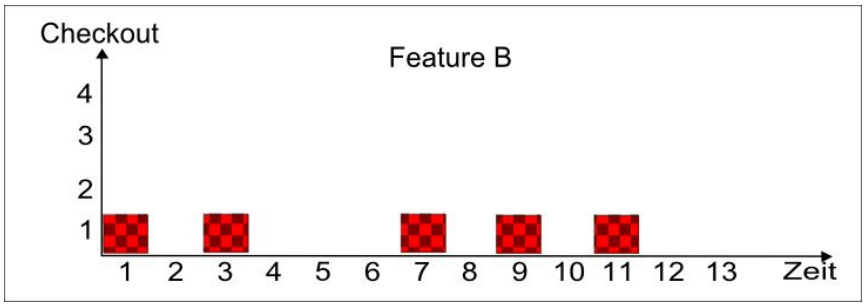

Abbildung 6.5: Beispiel für mögliches Auscheckverhalten von Feature B

Diese zwei Gebrauchssimulationen werden nun zu einer einzigen zusammengefasst (siehe 6.6). In dieser Teststrecke wird nun eine Ressourcenknappheit angenommen. Es kann nicht mehr als ein Feature zur gleichen Zeit ausgecheckt werden. Sollen zwei oder mehr Features ausgecheckt werden, werden diese in der Zeitachse nach hinten verschoben, bis dies möglich wird. Die Differenz zwischen dem Soll- und dem Ist-Checkoutzeitpunkt ergibt den Delay. Abbildung 5.11 zeigt diesen Vorgang.

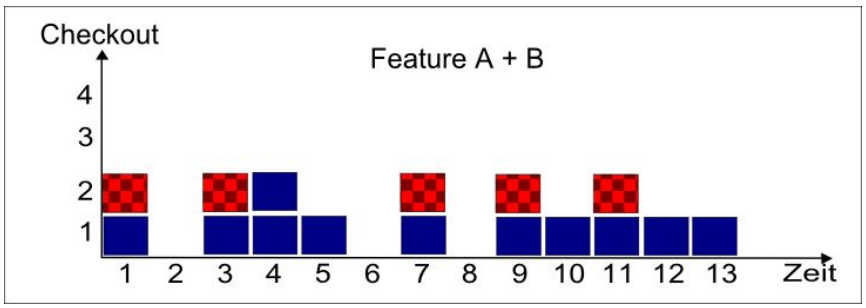

Abbildung 6.6: Kombination von von Auscheckverhalten A und B

6.4 Fazit

In diesem Kapitel wurde dargelegt, wie der theoretische Aufbau der durchgeführten Experimente aussieht. Es wurde erklärt, wie auf einfache Art und Weise gezeigt werden könnte, wie sich Sättigung und Segmentierung in Wechselwirkung miteinander verhalten. Fokus wurde hierbei auf die Minimierung der Einflussfaktoren gelegt, die unerwartete bzw. inkonsistente Ergebnisse liefern können. Die Anzahl der Features wurde auf zwei reduziert und die Einflussfaktoren Auscheckverteilung und Verteilung der Checkoutdauer fixiert. Da nun alle möglichen Einflussfaktoren neben Segmentierung und Sättigung eliminiert wurden, wurden diese nach Exhaustionsmethode (auch als Brute-Force-Methode bekannt) für alternierende Kombinationen getestet. Das folgende Kapitel 7 beschäftigt sich mit den Ergebnissen aus diesen Testreihen.

Kapitel 7

Interpretation der Ergebnisse aus den Simulationen mittels simulierter Daten

In diesem Kapitel werden die Ergebnisse aus den in Kapitel 6 durchgeführten Testläufen analysiert. Es wurden Daten für Gesamtsättigungen von 20 % bis 100 % für einen Beobachtungszeitraum von 3600 Zeiteinheiten angenommen. Für Feature A und B wurden alle möglichen Sättigungskombinationen in 10 % Schritten simuliert. Die Segmentierungsratio wurde von 1:1 bis hin zu 1:10 angehoben. Die Ergebnisse wurden visualisiert, analysiert und interpretiert.

7.1 Ergebnisse aus den Testreihen

Um einzelne Teststrecken miteinander vergleichen zu können, wurden diese in einzelne Gruppen unterteilt. Eine Gruppe wird definiert durch die Gesamtsättigung von A und B. Innerhalb einer solchen Gruppe werden nun alle möglichen Kombinationen aus Sättigungs- und Segmentierungsverhältnis als Datensatz eingetragen. Abbildung 7.1 zeigt wie ein Datensatz aussieht. In diesem Beispiel wird eine Segmentierungsratio zwischen A und B von 1 zu 1 angenommen. Waagrecht werden die einzelnen Kombinationen für Sättigung von A und B angegeben, die zusammen 80 % ergeben. Senkrecht wird die Segmentierung von A erhöht.

Wie man im Datenblatt erkennen kann, ist die Delayentwicklung bei steigender Segmentierung rückläufig. Unterscheiden sich die beiden Features in der Sättigung sehr, hat

Datenblatt für Gesamtsättigung 80% und Segmentierungsratio 1:1							
Segmentierung A	70%/10%	60%/20%	50%/30%	40%/40%	30%/50%	20%/60%	10%/70%
1	328	280	247	252	263	310	349
2	289	201	158	138	169	201	275
3	232	150	107	109	128	163	202
4	203	126	90	86	96	120	193
5	169	98	73	69	70	91	172
6	145	89	63	57	62	81	155
7	135	76	56	48	52	68	131
8	122	79	47	41	46	73	121
9	118	75	44	36	45	71	110
10	117	63	40	33	41	62	108
Summe	1858	1237	925	869	972	1240	1816

Abbildung 7.1: Delay für eine Segmentierungsratio von A:B = 1:1

dies weniger negative Auswirkungen auf den Delay, als wenn die Features gleich gesättigt sind. Summiert man die Delaywerte auf bzw. errechnet den Mittelwert, so erkennt man, dass der erwartete Delaywert für stark unterschiedliche Features drei mal so hoch ist. Die Werte für die Sättigungsratio 70 % zu 10 % und 10 % zu 70 % sind in diesem Datenblatt ähnlich aufgrund der gleichen Segmentierung. Um Datensätze besser analysieren und vergleichen zu können, wurden sie auf einem Verlaufsdiagramm aufgetragen. Abbildung 7.2 zeigt das erstellte Verlaufsdiagramm für die in Abbildung 7.1 vorgestellten Daten.

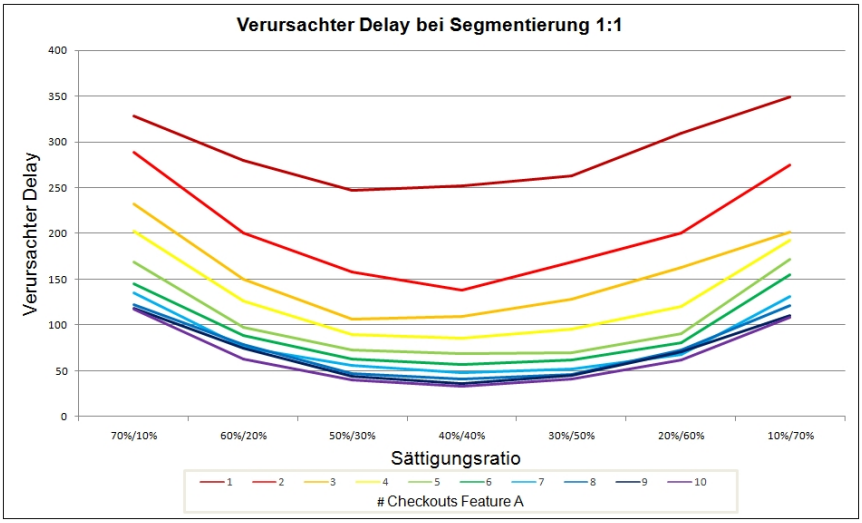

Abbildung 7.2: Verlaufsdiagramm der Delayentwicklung für Segmentierungsratio A:B = 1:1 wobei Segmentierungswerte von A = 1-10

Dieser Vorgang wurde nun ebenfalls für weitere Segmentierungsverhältnisse durchgeführt. Die ermittelten Verlaufsdiagramme wurden gegenüber gestellt und verglichen. Abbildung 7.3 zeigt dies für die folgenden Segmentierungsverhältnisse: 1:1, 1:2, 1:3 und 1:9.

Wie man aus diesen Verlaufsdiagrammen erkennen kann, ist die Delayentwicklung bei

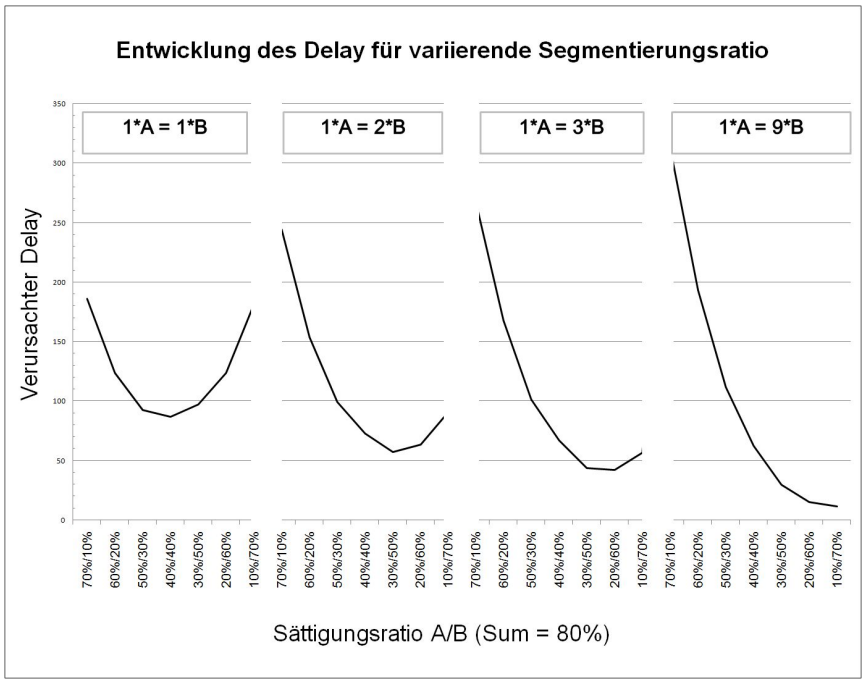

Abbildung 7.3: Verlaufsdiagramme der Delayentwicklung für Segmentierungsratio A:B = 1:1 bis 1:9

variierender Segmentierungsratio nicht wie erwartet. Vergleicht man jeweils die Werte für 70 % zu 10 %, stellt man eine Steigerung des Delay fest, trotz der insgesamt höheren Segmentierung. Im umgekehrten Fall, 10 % zu 70 %, verringert sich der Delay. In weiterer Folge wurde versucht eine Gesamtübersicht über die Entwicklung der beiden Variablen Segmentierung und Sättigung zu erstellen. Diese Gesamtübersicht soll alle Datenblätter in einer Grafik vereinen. Das Ergebnis ist Abbildung 7.4.

Betrachtet man diese Abbildung könnte man annehmen, dass Segmentierung keinerlei Einfluss auf den Delay hat. Mit sinkender Sättigung sinkt auch der verursachte Delay. Durch Abbildung 7.3 kann man jedoch erkennen, dass dies nicht der Fall ist. Im folgenden Abschnitt werden die Ergebnisse interpretiert.

7.2 Interpretation der Ergebnisse

In diesem Abschnitt wird näher auf die Entwicklung des Delay in bestimmten Szenarien eingegangen. Das erste Szenario wird in Abbildung 7.5 dargestellt. Wie man erkennen kann, wirkt Feature A (hohe Sättigung, geringe Segmentierung) blockierend für das kurzlebigere Feature B. Dieses Feature hat zwar eine kürzere Auscheckdauer, aber die Segmentierung

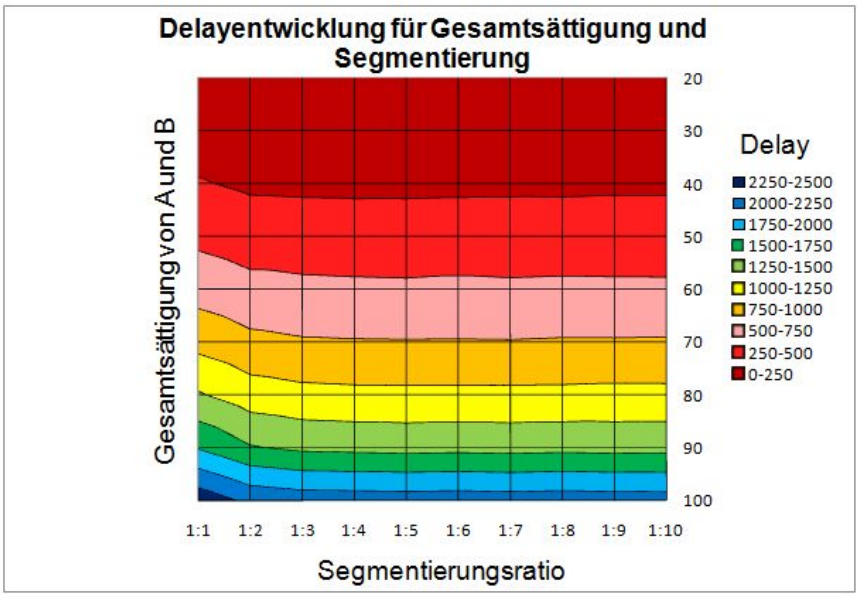

Abbildung 7.4: Gesamtansicht für alle Ergebnisse aus Segmentierungs- und Sättigungskombinationen als Flächendiagramm

ist ebenfalls sehr gering. Dies hat zur Folge, dass der Delay für steigende Segmentierung abfällt. Nimmt man die Ergebnisse aus Abbildung 7.1 zur Hand entspräche dieses Szenario der Teststrecke für eine 40 % zu 40 % oder 50 % zu 30 % Sättigungsrelation. Aufgrund des ähnlichen Sättigungsverhältnisses gibt es hier eine stabile Delayentwicklung bei der der Tausch von A und B ohne Auswirkung bliebe. Für einen Tokenpool wären dies die perfekten Kandidaten, da sie ähnliche Eigenschaften besitzen und somit leicht, ohne Nebeneffekte, berechen- und austauschbar sind.

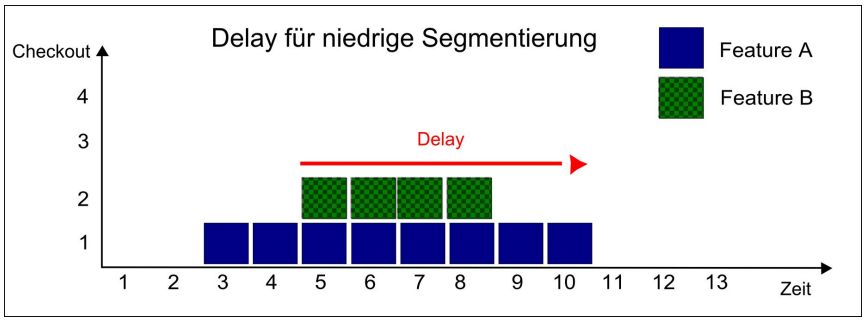

Abbildung 7.5: Szenario 1 für Delayverursachung: 2 ähnliche Features

Abbildung 7.6 zeigt nun das selbe Szenario, mit dem Unterschied, dass Feature B durch eine höhere Segmentierung nun weiter aufgespalten ist. Durch die höhere Anzahl der Check-

outs kann nun Feature B, unabhängig von der Sättigung, mehr Delay verursachen. Dieses Szenario kann man auf Abbildung 7.3 sehr gut erkennen. Vergleicht man jeweils den Delay für Sättigungsratio 70 % zu 10 %, so kann man erkennen, dass der Delay mit steigender Segmentierung ebenfalls steigt. In diesem Fall verhält sich Feature A wie eine Blockade für alle anderen Features die ebenfalls zu diesem Zeitpunkt ausgecheckt werden wollen. Je mehr Features das sind, desto höher summiert sich der Delay auf.

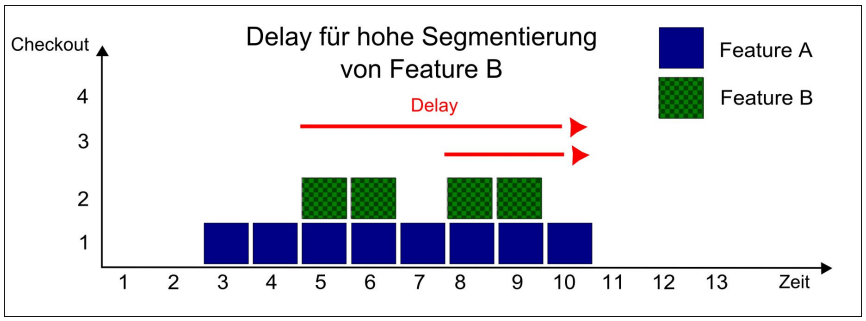

Abbildung 7.6: Szenario 2 für Delayverursachung: Feature A mit hoher Sättigung und geringer Segmentierung und Feature B mit niedriger Sättigung und hoher Segmentierung

Die folgende Abbildung 7.7 veranschaulicht, wie der umgekehrte Fall aussieht. Feature A besitzt nach wie vor eine geringe Segmentierung, während Feature B eine hohe hat. Die Sättigung ist allerdings vertauscht. Da Feature A nicht mehr als Blockade effektiv ist und Feature B stark verteilt ist, sinkt der Delay. In Abbildung 7.3 entspricht dieses Szenario den Werten für die 70 % zu 10 % Sättigungsratio. Erhöht man die Segmentierungsratio für ein solches Szenario, so verringert sich der Delay.

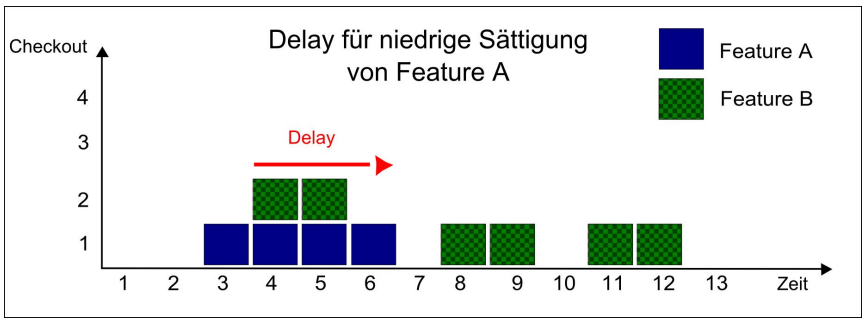

Abbildung 7.7: Szenario 2 für Delayverursachung: Feature A mit niedriger Sättigung und geringer Segmentierung und Feature B mit hoher Sättigung und hoher Segmentierung

Wie man anhand dieser Szenarien erkennen kann, ist der Einfluss, der von der Segmentierung ausgeübt wird, daran gekoppelt, wie die betreffenden Features gesättigt sind.

Es wurde gezeigt, dass die Wirkung einer Steigerung der Segmentierung, je nach Fall, entgegengesetzte Wirkung hat (siehe 7.6 und 7.7).

7.3 Visualisierung der Daten aufgrund der Interpretation

Abbildung 7.4 zeigt, dass die Delayentwicklung von der Gesamtsättigung abhängig ist. Aus dieser Abbildung lassen sich jedoch keine Rückschlüsse auf den Einfluss der Segmentierungsratio ziehen. Wie in Abschnitt 7.2 festgestellt, kann dies dennoch große Auswirkung auf einzelne Szenarien haben. Um nun diese Effekte in Anwendungsbeispielen berücksichtigen zu können, wurden Grafiken erstellt, für die Sättigung und Segmentierung aneinander gekoppelt wurden. Die Abbildungen 7.8 bis 7.14 zeigen die Darstellungen für die Sättigungsratio 1:1, 1:2, 1:3, 2:1, 2:3, 3:1 und 3:2.

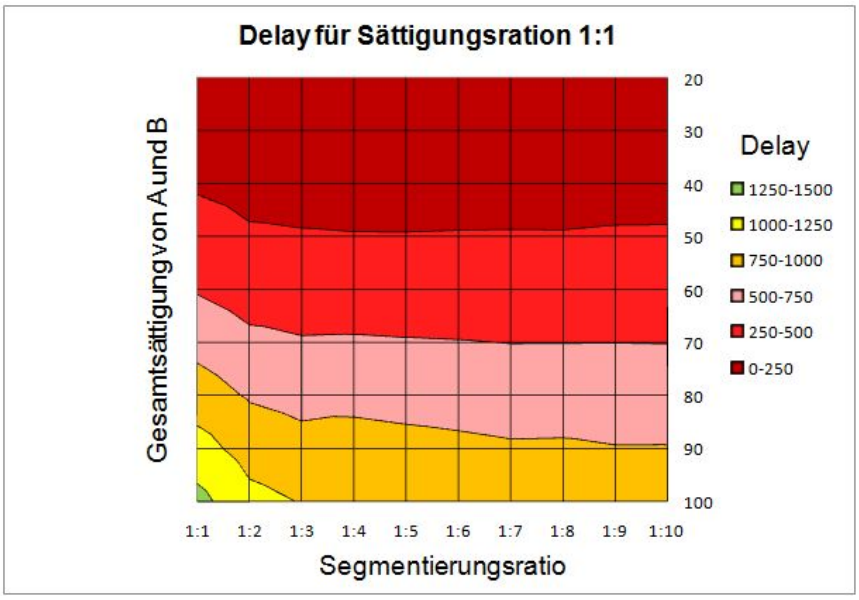

Abbildung 7.8: Ermittelte Delayentwicklung für Sättigungsratio 1:1

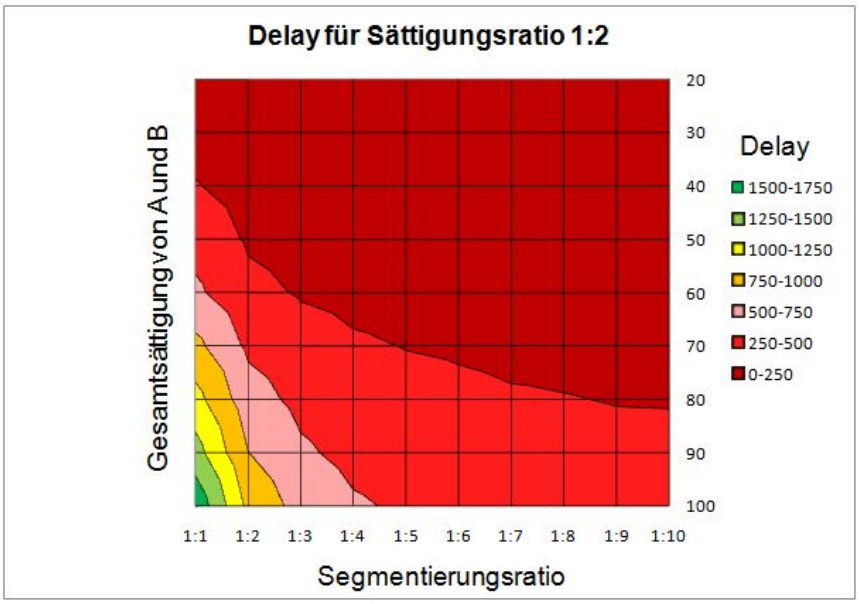

Abbildung 7.9: Ermittelte Delayentwicklung für Sättigungsratio 1:2

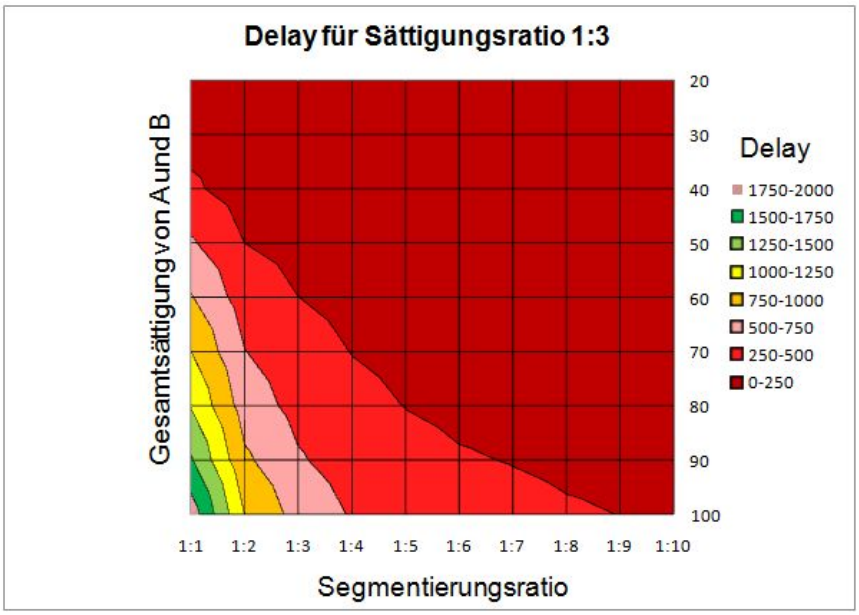

Abbildung 7.10: Ermittelte Delayentwicklung für Sättigungsratio 1:3

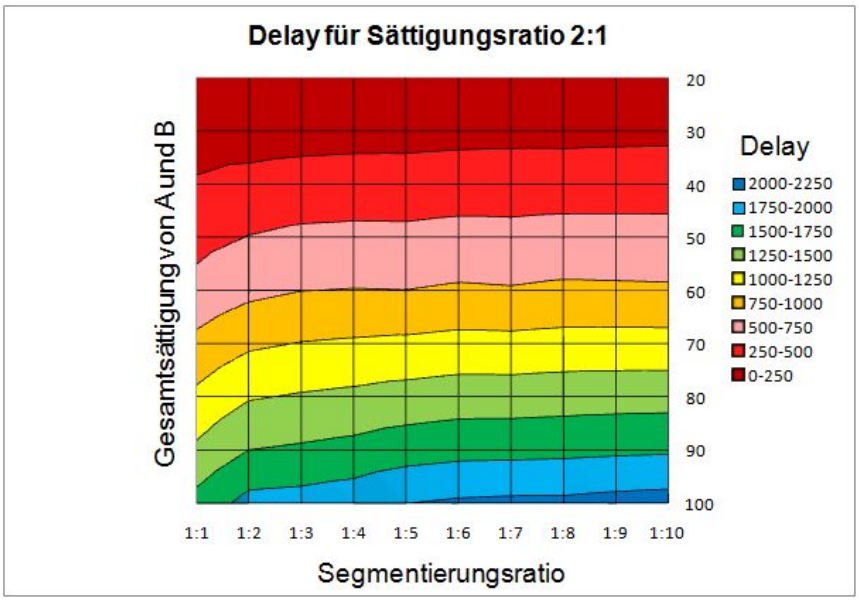

Abbildung 7.11: Ermittelte Delayentwicklung für Sättigungsratio 2:1

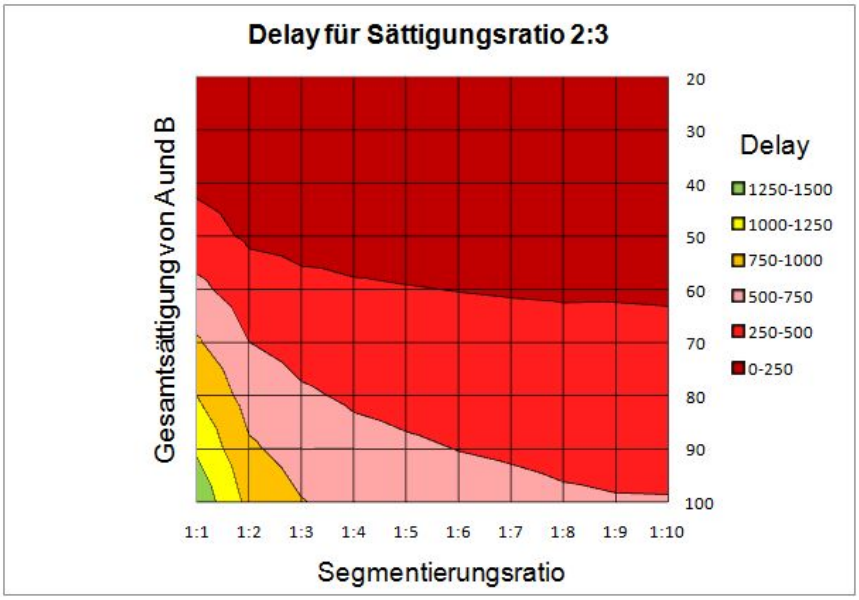

Abbildung 7.12: Ermittelte Delayentwicklung für Sättigungsratio 2:3

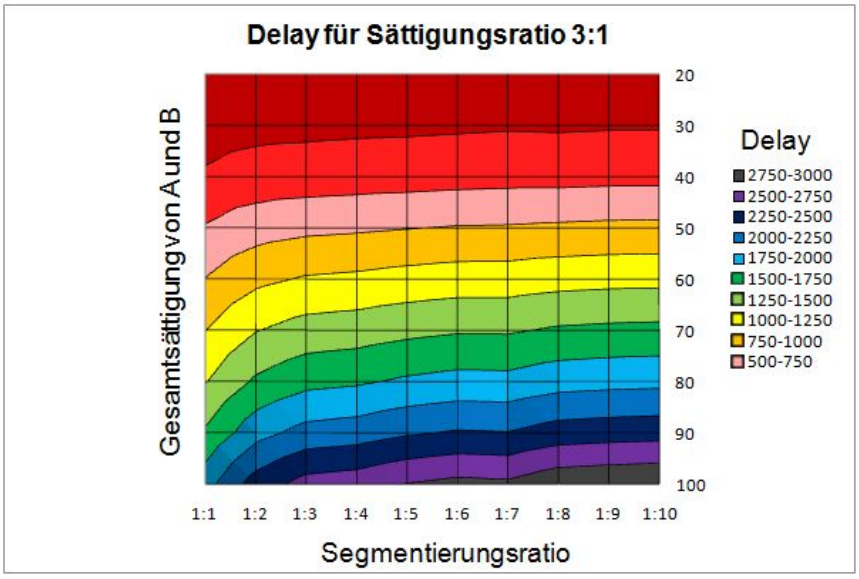

Abbildung 7.13: Ermittelte Delayentwicklung für Sättigungsratio 3:1

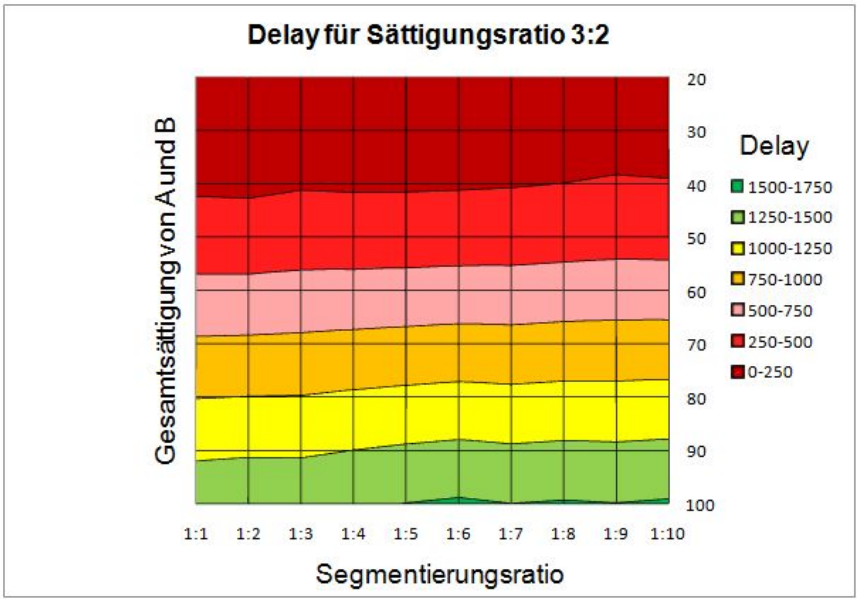

Abbildung 7.14: Ermittelte Delayentwicklung für Sättigungsratio 3:2

7.4 Anwendungsbeispiele

In diesem Abschnitt wird erläutert, wie die in Abschnitt 7.1 und 7.3 ermittelten Daten und erstellten Grafiken angewandt werden können, um nun Tokenpools zu errechnen. Dabei wird eine Anzahl von Features angenommen, die in einem Tokenpool miteinander kombiniert werden sollen. Von diesen Features sind die Eigenschaften Sättigung und Segmentierung bekannt. Nun werden die Einträge aus den Lookup-Tabellen aus Kapitel 7.3 ermittelt und in eine Formel eingesetzt. Die Formel, die hier angewandt wird, kann individuell ermittelt werden. Sie gibt an, wie der Delay mit den anderen Eigenschaften der Features gewertet wird. Im Zuge dieser Demonstration, führt eine einfache Multiplikation von Delay (aus den Lookup-Tabellen), Sättigung und Segmentierung zu den gewünschten Tokenwerten.

7.4.1 Beispiel 1

Für Beispiel 1 wird das einfachste Szenario angenommen: 2 Features, A und B, mit bekannter Sättigung und Segmentierungsratio.

Eigenschaften Feature A:

- Sättigung: 60 %
- Segmentierung: 1

Eigenschaften Feature B:

- Sättigung: 20 %
- Segmentierung: 5

Abbildung 7.15 zeigt einen Ausschnitt aus den Daten für eine 3:1 Sättigungsratio. Ließt man den entsprechenden Wert für die jeweiligen Sättigungs- und Segmentierungsratio ab, erhält man als Ergebnis 1787. Da die Normierung jetzt noch aussteht, wird 1787 in diesem Beispiel auf 100 % gesetzt. Um nun die einzelnen Werte miteinander in Abhängigkeit zu bringen, kann man, beispielsweise, Sättigung, Segmentierung und Delay miteinander multiplizieren. Das Produkt kann in weiterer Folge als Grundlage für die Bewertung der Features herangezogen werden.

Nach dieser Rechnung ist das Ergebnis für Feature A gleich 0.6 (1*0.6*1.00) sowie 1.0 (5*0.2*1.00) für Feature B.

Ausschnitt aus Lookup-Tabelle für unterschiedlich Gesamtsättigung und Segmentierungsratio			
	Gesamtsättigung		
Segmentierungsratio	100%	90%	80%
1:1	1903,27	1534,38	1237
1:2	2367,96	1908,38	1538
1:3	2594,19	2087,01	1679
1:4	2642,96	2129	1715
1:5	2758,7	2220,31	1787
1:6	2822,3	2273,86	1832
1:7	2797,79	2259,02	1824
1:8	2933,73	2361,57	1901
1:9	2964,34	2388,79	1925
1:10	2985,22	2405,27	1938

Abbildung 7.15: Ausschnitt aus den Daten für 3:1 Sättigungsratio

Dies kann nun für eine beliebige Anzahl von Features durchgeführt werden. Hat man dann alle Features bewertet, können Tokenwerte ermittelt werden, indem eine Basis für alle Features festgelegt und daraufhin mit dieser multipliziert wird. Für dieses Beispiel wird eine Basis von 1000 angenommen. Mit dieser Basis kommt man nun auf die Tokenzahlen:

- Token für Feature A: 600
- Token für Feature B: 1000

Will man nun eine Grenze für den Tokenpool festlegen, wird in weiterer Folge eine Monte Carlo Simulation für diese Tokenwerte durchgeführt. Diese laufen gleich ab wie in Abschnitt 4.7 beschrieben.

Monte Carlo Simulation			
#Testlauf	A	B	Sum
1	0	0	0
2	600	0	600
3	600	0	600
4	0	0	0
5	0	0	0
6	1200	0	1200
7	600	1000	1600
...	...	...	...
...	...	...	...
1000	0	0	0

Abbildung 7.16: Ausschnitt aus der Monte Carlo Simulation für Bsp. 1

Aus dieser Monte Carlo Simulation (siehe Abbildung 7.16) lässt sich die Anzahl der Ereignisse errechnen, für die eine bestimmte Anzahl an Token verwendet wird. Daraus lässt sich der Bedarf für Token ableiten und man kann daraufhin bei einem bestimmten %-Wert festgelegt werden. Tabelle 7.17 soll dies veranschaulichen.

Tokennutzung	
Anzahl gebrauchter Tokens	Ereignisse (% - akkumuliert)
600	50,2
1000	54,3
1600	60,5
2000	61
2600	62,1
0	1

Abbildung 7.17: Akkumulierte Tokennutzung für Bsp. 1

Abbildung 7.18 zeigt die in Tabelle 7.17 aufgeführten Daten. Zwischen 600 Token und 1600 Token steigt der Bedarf an, während der weitere Verlauf (von 1600 bis 2000) weniger stark ausfällt. D.h. eine Erhöhung des Tokenpools von 600 auf 1000 (bzw. von 1000 auf 1600) hat einen größeren Mehrwert für einen Kunden, als eine Erhöhung von 1600 auf 2000.

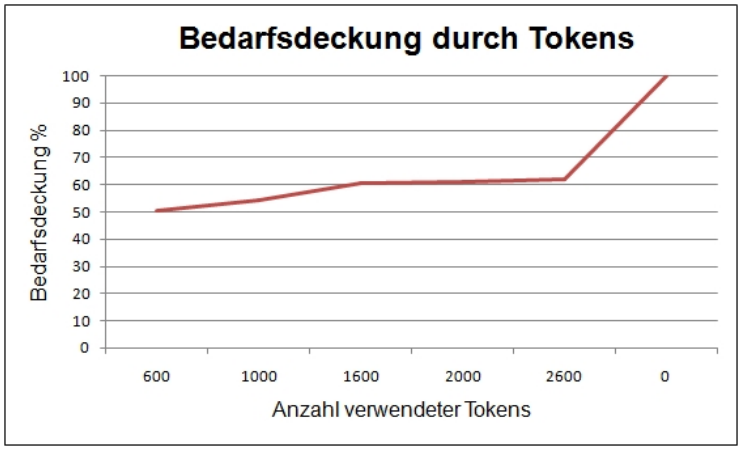

Abbildung 7.18: Bedarfsdeckung durch Token für Rechenbeispiel 1

7.4.2 Beispiel 2

Hier wird nun ein Beispiel für drei Features präsentiert. Da in den Ergebnistabellen für Delay nur Featuretupel ablesbar sind, müssen hier mehrere Abhängigkeiten zwischen den Features berücksichtigt werden. Für die Features A, B und C müssen folgende Relationen erwogen werden: AB, AC und BC. Die Angaben für Feature A und B sind aus dem vorangegangen Beispiel beibehalten worden. Feature C hat 30 % Sättigung und folgende Relationen zu A und B (siehe Abbildung 7.19).

Sättigungsrelation			
	Feature A	Feature B	Feature C
Feature A	1:1	3:1	2:1
Feature B	3:1	1:1	2:3
Feature C	2:1	2:3	1:1

Abbildung 7.19: Sättigungsrelationen für Beispiel 2

Um nun den Delay richtig ablesen zu können, muss für jede Einzelrelation, AB, AC und BC, auch die Segmentierungsratio vorhanden sein. Diese ist für dieses Beispiel in Matrixform (siehe Abbildung 7.20).

Segmentierungsratio			
	Feature A	Feature B	Feature C
Feature A	1:1	1:5	1:1
Feature B	1:5	1:1	1:5
Feature C	1:1	1:5	1:1

Abbildung 7.20: Segmentierungsrelationen Beispiel 2

Liest man nun für jede Relation die Delaywerte aus den entsprechenden Datenblättern ab, erhält man folgende Werte (siehe Abbildung 7.21).

Delay (aus LookUp-Tabelle)			
	Feature A	Feature B	Feature C
Feature A	x	1787	1292
Feature B	1787	x	167
Feature C	1292	167	x

Abbildung 7.21: Ergebnisdelay für die Featurerelationen aus Beispiel 2

Die Normierung erfolgt (siehe Abbildung 7.22). 1787 wird gleich 1 gesetzt.

Normierter Delay			
	Feature A	Feature B	Feature C
Feature A	x	1,00	0,72
Feature B	1,00	x	0,09
Feature C	0,72	0,09	x

Abbildung 7.22: Normierter Ergebnisdelay für die Featurerelationen aus Beispiel 2

In weiterer Folge werden wieder Sättigung, Segmentierung und Delay miteinander multipliziert und der Mittelwert aus den Ergebnissen gezogen. Abbildung 7.23 veranschaulicht diesen Vorgang.

Rechenvorgang				
	Feature A	Feature B	Feature C	Mittel
Feature A	x	1,0*0,6*1,0 = 0,60	1,0*0,6*0,72 = 0,43	0,51
Feature B	5,0*0,2*1,0 = 1,00	x	2,0*0,2*0,9=0,36	0,68
Feature C	1,0*0,3*0,72 = 0,21	3,0*0,3*0,9 = 0,81	x	0,51

Abbildung 7.23: Rechenvorgang für die Errechnung von Tokenwerten für Beispiel 2

Die Ergebnisse für Feature A, B und C sind 51 %, 68 % und 51 %. Dies ergibt, multipliziert man diese Werte mit dem Grundtokenwert von 1000, für A und C 510 Token und für B 680 Token. Wendet man mit diesen Werten ebenfalls eine Monte Carlo Simulation an, bekommt man folgenden Ergebnisgraph (siehe Abbildung 7.24).

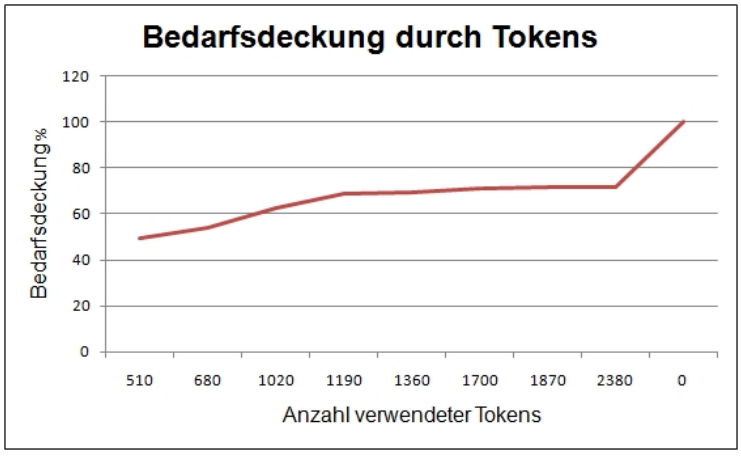

Abbildung 7.24: Bedarfsdeckung durch Token für Rechenbeispiel 2

Will man hier z.B. einen Grenzwert für den Tokenpool von A, B und C bei etwa 60 % festlegen, wäre er nach Monte Carlo Simulation bei 1200 Token. Die dargestellte

Linie ist im Gesamtverlauf flacher. Eine größere Anzahl an Kombinationsmöglichkeiten an Tokenwerten ist aufgrund der höheren Anzahl der Features möglich. Dies hat zur Folge, dass die Auswahlmöglichkeiten an sinnvollen Tokenpools größer ist.

7.5 Zusammenfassung und Fazit

Mit den in diesem Kapitel vorgestellten Lookup-Tabellen ist es möglich Features miteinander zu kombinieren und deren individuelle Eigenschaften, sowie die Interaktion, die diese miteinander haben, auf einfache Art und Weise zu bewerten. Der Vorgang hierfür ist wie folgt:

- auswählen der Features die miteinander kombiniert werden sollen,
- feststellen der Eigenschaften der Features,
- feststellen des Interaktionsverhaltens der Features,
- ablesen des Delay in den Lookup-Tabellen und
- Anwendung des Tokenbewertungsalgorithmus.

Mittels dieser Schritte lassen sich auf einfache Weise Tokenwerte für die gewünschten Features ermitteln. Hierbei ist mit in Betracht zu ziehen, dass mit steigender Anzahl an Features auch die Ungenauigkeit stark anwächst. Ein Token basierendes Lizensierungsmodell ist, aufgrund seiner hohen Dynamik, nicht auf eine hohe Featurezahl ausgelegt. Ein weiterer Unsicherheitsfaktor ist der Mensch. Eine Verteilung der Checkouthäufigkeit und Checkoutdauer kann eine zufällige Benutzung durch den Menschen nicht 100 % widerspiegeln. Hinzu kommt noch, dass ein Missbrauch eines solchen dynamischen Systems durch Optimierungssoftware oder andere Möglichkeiten niemals auszuschließen ist. Für die Zukunft könnte eine solche Lookup-Tabelle jedoch Anwendung finden, indem sie eine erste Annäherung bzw. einen Richtwert für weitere Überlegungen bietet. Sättigung, Segmentierung, Auscheckverteilung und Auscheckdauer sind, wie in Kapitel 5 beschrieben, nicht die einzigen Faktoren, die eine Rolle spielen können. Anzumerken ist, dass im Rahmen dieser Arbeit nur ein Interaktionsmodell, das der Gleichverteilung, näher behandelt wurde. Für weitaus genauere Annäherungen an eine akkurate Tokenbewertung, können noch Anwendungssimulationen für weitere Verteilungen (wie in Abschnitt 5.1.2 vorgestellt) gemacht werden.

Kapitel 8

Zusammenfassung

Die Abteilung AST (Advanced Software Technology) der Firma AVL entwickelt für den Eigengebrauch Simulationssoftware. Diese Software wird für unterschiedliche Phasen in der Automobilentwicklung angewandt und besteht aus vier Paketen. Diese Softwarepakete sind weiter in mehrere Einzellösungen unterteilt. Diese Softwarepakete werden ebenfalls zum Verkauf angeboten. Dies geschieht über ein Lizensierungssystem das auf Zeitbasis (jährlich) arbeitet. Dieses Projekt wurde im Auftrag der Firma AVL gestartet, mit dem Ziel, deren bestehendes Lizensierungssystem zu verbessern. Kunden der Firma haben den Wunsch nach dynamischerer Lizenzverwaltung ausgedrückt. Ebenfalls ist der Firma aufgefallen, dass gewisse Softwarelösungen sehr selten Verwendung finden, und daher nicht häufig lizensiert werden. Dies gab den Anstoß für den Auftrag, alternative Lizensierungsmethoden zu erforschen. In dieser Arbeit wurde ein Überblick über die Möglichkeiten für Softwarelizensierung gegeben. Es wurde auf 'State of the Art' Technologien und Methoden aufmerksam gemacht. Der erste Teil ermöglicht einen Einblick in die Theorie, welche einen Überblick über Einflussfaktoren und Herangehensweise an den Sachverhalt liefert. Im zweiten Teil wird ein näherer Blick auf die Lizensierungssysteme der direkten Konkurrenten der Firma AVL geworfen, sowie erfolgreiche andere Unternehmen mit dynamischen Lizensierungsarten. Nachdem mehrere Lizensierungsarten vorgestellt wurden, fiel die Entscheidung, eine Token Based Lizensierungsstrategie zu realisieren. Um dies zu bewerkstelligen wurden Logfiles vom Lizensierungsserver der AVL bereit gestellt. Vorgabe war, ein Token Based Setup zu finden, das möglichst geringen Einfluss auf die Einnahmen der Firma hat, im Vergleich mit dem alten Lizensierungssystem. Es wurde versucht die wichtigen Daten aus den Logfiles zu extrahieren und für Gebrauchssimulationen anzuwenden. Diese Daten bestanden aus Benutzerverhalten von einzelnen Mitarbeitern der AVL und einiger ausge-

wählter Features. Die Features wurde nach Zeit, Auscheckdauer und paralellen Checkouts geparsed. Nachdem die Features nach diesen Eigenschaften analysiert waren, wurde ihnen ein beliebiger Tokenwert zugewiesen. Danach wurde eine Vielzahl an unterschiedlichen Kombinationen für Tokenwerte nach der Brute-Force-Methode getestet. Es wurden Bewertungsgrößen ermittelt, die es ermöglichen, verschiedene Kombinationen aus Tokenwerten zu vergleichen. Diese Bewertungsgrößen wurden auch für die alten Netzwerklizenzen ermittelt. Aus diesen Testreihen ging hervor, dass der Versuch mittels Brute-Force-Methode ein zuverlässiges und schlüssiges Ergebnis zu erzielen, gescheitert war. Ein neues Konzept für die Erstellung von Token Based Lizensierungsverfahren wurde erarbeitet. Das neue Konzept sieht vor, Features anhand seiner Eigenschaften gezielt bewerten zu können. Es sollte nicht mehr ein Quasioptimum mittels Brute-Force-Methode ermittelt werden, sondern eine fest definierte Formel, die anhand verschiedener Inputgrößen ein eindeutiges Ergebnis liefert. Die Eigenschaften von Features und deren Interaktion miteinander wurde als Inputgröße herangezogen. Mittels dieser Inputgrößen wurde ermittelt, welcher Delay nach Interaktion dieser Features zu erwarten ist. Dieser Delay wird in die Formel eingebaut und stellt eine Relation zwischen Sättigung und Segmentierung zweier Features her. Ist diese Relation hergestellt, kann das Feature anhand dieser Größen bewertet werden.

8.1 Ausblick

Ein Weg zur Erstellung von Lookup-Tabellen für Features mit unterschiedlichen Eigenschaften ist das Ergebnis, das aus dieser Arbeit hervor ging. Die in diesem Dokument präsentierten Tabellen und Grafiken stellen jedoch nur einen kleinen Teil der Möglichkeiten, die in einem Softwareportfolio vorkommen können, dar. Für einen weiterführenden Blick auf die Eigenschaften und das Verhalten unterschiedlichster Software sind vor allem noch unterschiedlichste Arten der Interaktion zwischen Features zu untersuchen. Im konkreten Fall der Firma AVL wurde die Empfehlung gegeben, dass die Umsetzung eines Token Based Lizensierungsmodells nicht rentabel wäre. Der Einflussfaktor der Checkouthäufigkeit vieler der Features im Portfolio der AVL war zu unterschiedlich. Es wurde das Risiko eines Missbrauchs von Seiten der Kunden für zu hoch eingeschätzt, als das sich eine Umsetzung rentieren würde. Zusätzlich wäre durch eine starke Polarisierung der Tokenwerte für Features das Gegenteil zum eigentlich gewünschten Effekt aufgetreten. Der wäre, dass das Token Based Lizensierungsmodell als Anreizsystem dient, um dem Kunden den Kauf weiterer Software schmackhaft zu machen. Sehr unterschiedliche Tokenwerte laden indes

zu einer gegenteiligen, minimalistischen Handhabung ein. Zusammenfassend waren also die sehr hohe Anzahl der zu bewertenden Features, mit der damit einhergehenden Komplexität der Berechnung, sowie die sehr unterschiedlichen Werte für Segementierung, Sättigung, Auscheckdauer und Auscheckhäufigkeit der Grund für das Beibehalten des ursprünglich in Verwendung befindlichen Lizensierungsmodells.

Literaturverzeichnis

[AB07] Fred Andresen and Ulrich Bantle. Fallstricke im End User License Agreement. *Linux-Magazin*, 03, 2007. http://www.linux-magazin.de/Heft-Abo/Ausgaben/2007/03/Ich-seh-dich-nicht (Letzter Besuch: 06.02.2012).

[AVL08] AVL. *AVL PRODUCT DESCRIPTION BOOST*. https://www.avl.com/web/ast/boost (Letzter Besuch: 15.11.2011), 2008. Abteilung: Advanced Simulation Technologies.

[AVL09] AVL. *AVL FIRE General Purpose*. https://www.avl.com/web/ast/fire (Letzter Besuch: 15.11.2011), 2009. Abteilung: Advanced Simulation Technologies.

[AVL10a] AVL. *AVL BOOST - Linear Acoustics*. https://www.avl.com/web/ast/boost (Letzter Besuch: 15.11.2011), 2010. Abteilung: Advanced Simulation Technologies.

[AVL10b] AVL. *AVL PRODUCT DESCRIPTION BOOST AFTERTREATMENT*. https://www.avl.com/web/ast/boost (Letzter Besuch: 15.11.2011), 2010. Abteilung: Advanced Simulation Technologies.

[AVL11a] AVL. *AVL Excite*. https://www.avl.com/web/ast/excite (Letzter Besuch: 15.11.2011), 2011. Abteilung: Advanced Simulation Technologies.

[AVL11b] AVL. *AVL Product Description Cruise*. https://www.avl.com/web/ast/boost/cruise (Letzter Besuch: 15.11.2011), 2011. Abteilung: Advanced Simulation Technologies.

[Csi09] Zsigri Csilla. THE BUSINESS SIDE OF SOFTWARE LICENSING. Technical report, THE 451 GROUP, http://www.451group.com/ (Letzter Besuch: 06.02.2012), 2009. SmartLM Projekt.

[CTW98] A. Chavez, C. Tornabene, and G. Wiederhold. Software component licensing: a primer. *IEEE Software*, 15(5):47–53, 1998.

[DB10] Steve Duscheid and Randy Britton. Tech Talk # 1 - Software Usage Analysis : A key to IT cost savings. (June 2006):1–23, 2010.

[DG] V. D'Andrea and G.R. Gangadharan. LicensingWeb Services: The Rising. *Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services (AICT-ICIW'06)*, pages 142–142.

[DP09] Mathias Dalheimer and Franz-Josef Pfreundt. GenLM: License Management for Grid and Cloud Computing Environments. *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 132–139, 2009.

[GD06] G R Gangadharan and Vincenzo D D'Andrea. Licensing Services : Formal Analysis and Implementation. pages 365–377, 2006.

[GW09] Daniel Gull and Alexander Wehrmann. Optimized Software Licensing ? Combining License Types in a License Portfolio. *http://www.wirtschaftsinformatik.de*, 2009.

[Idr04] Kamil Idris. Successful Technology Licensing. IP ASSETS MANAGEMENT SERIES 903E, World Intellectual Property Organization, http://www.wipo.int/ip-development/en/strategies/pdf/publication_903.pdf (Letzter Besuch: 06.02.2012), 2004.

[Jon10] Mun Jonathan. *Modeling risk: applying Monte Carlo risk simulation, strategic real options, stochastic forecasting, and portfolio optimization.* John Wiley & Sons, Inc, Hoboken, New Jersey, 2 edition, 2010.

[KP05] Halina Kaminski and Mark Perry. The Pattern Language of Software Licensing. *Identity, europlop2005*, pages 1–41, 2005.

[Lic11a] Licence Tracker. License Tracker in Utility Computing and Application Service Provider Operations. Technical report, License Tracker Inc.(LTI), http://www.licensetracker.ca/ (Letzter Besuch: 06.02.2012), 2011.

[Lic11b] Licence Tracker. The Evolution of Software Licensing Models. Technical report, License Tracker Inc.(LTI), http://www.licensetracker.ca/ (Letzter Besuch: 06.02.2012), 2011.

[LZWM08] Jiadao Li, Wolfgang Ziegler, Oliver Waeldrich, and Daniel Mallmann. *Towards SLA Based Software License Management in Grid Computing.* PhD thesis, Institute on Resource Management and Scheduling, 2008.

[Mac03] Corporation Macrovision. OVERCOMING THE SOFTWARE LICENSING COMPLEXITY CRISIS - The Case for a Universal Licensing Platform. 2003.

[Mac06] Macrovision. Flexnet Licensing End User Guide. *Product Version 11.4 Document revision 01*, 2006.

[SG04] Laurie A Seymour and Stephen Graham. The Future of Software Licensing : Software Licensing Under Siege. 2004.

[SIL10a] SILVACO. *TCAD TOKENS*. http://www.silvaco.com/licensing/licensing.html (Letzter Besuch: 15.11.2011), 2010. Produktportfolio TCAD.

[SIL10b] SILVACO. *TOKEN BASED*. http://www.silvaco.com/licensing/licensing.html (Letzter Besuch: 15.11.2011), 2010. Remix Licensing.

[SIL12] SILVACO. *UNIVERSAL TOKENS - For all TCAD and EDA tools*. http://www.silvaco.com/licensing/universal/universal_token.html (Letzter Besuch: 05.02.2012), 2012. Produktportfolio.

[SWZ07] Jan Seidel, Oliver Waeldrich, and Wolfgang Ziegler. Using SLA for resource management and scheduling - a survey. Survey, CoreGRID Technical Report - Institute on Resource Management and Scheduling, http://www.coregrid.net (Letzter Besuch: 06.02.2012), 2007.

[TJ05] C.W. Thompson and R. Jena. Digital Licensing. *IEEE Internet Computing*, 9(4):85–88, Juli 2005.

[TKK09] Timo Tuunanen, Jussi Koskinen, and Tommi Kärkkäinen. Automated software license analysis. *Automated Software Engineering*, 16(3-4):455–490, 2009.

Abbildungsverzeichnis