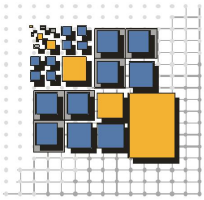

Otto-Friedrich-Universität Bamberg

Lehrstuhl für Praktische Informatik

Bachelorarbeit

im Studiengang Angewandte Informatik
der Fakultät Wirtschaftsinformatik und Angewandte Informatik
der Otto-Friedrich-Universität Bamberg

Zum Thema:

Klassifikation und Vergleich von Preismodellen für Cloud-Plattformen

Vorgelegt von:

Thomas Aberle

Themensteller:
Prof. Dr. Guido Wirtz

Abgabedatum:
01.07.2017

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

IaaS Infrastructure-as-a-Service

PaaS Platform-as-a-Service

SaaS Software-as-a-Service

1 Einleitung

Cloudanwendungen sind heutzutage aus der Businesswelt nicht mehr wegzudenken. Sowohl Anwendungen als auch Daten werden mittlerweile mehr und mehr in Cloudsysteme verschiedenster Anbieter ausgelagert, wobei Systeme, welche in die Kategorie *Cloud* fallen, in unterschiedlichsten Ausprägungen vorkommen können. Cloudsysteme lassen sich, basierend auf ihrem Abstraktionsgrad, in unterschiedliche Servicekategorien einteilen. Es wird grundsätzlich zwischen den Modellen *Infrastructure-as-a-Service (IaaS)*, *Platform-as-a-Service (PaaS)* und *Software-as-a-Service (SaaS)*(vgl. [1]) unterschieden.
Der IaaS-Bereich stellt die unterste Ebene des Cloud-Stacks dar. Dem Nutzer wird vom Anbieter lediglich die Hardware- und Netzwerkinfrastruktur bereitgestellt. Der Rest bleibt dem Nutzer selbst überlassen, inklusive der Wahl des Betriebssystems, der Middleware und der Anwendungen, die verwendet werden sollen.
In der PaaS-Kategorie, der mittleren Ebene des Cloud-Service-Models, stellt der Anbieter dem Nutzer nicht nur die Infrastruktur aus Hardware und Netzwerk. Zusätzlich werden auch Betriebssystem, Middleware und Laufzeitumgebungen, den Wünschen des Nutzers entsprechend, geliefert.
Die dritte und letzte Cloud-Ebene bildet die SaaS-Kategorie. Hier wird dem Anwender eine vollständige Cloudanwendung bereitgestellt, die dieser ohne großen Konfigurationsaufwand verwenden kann.

Während IaaS bereits mehr und mehr standardisiert wird, stellt sich dies bei PaaS- und SaaS-Systemen als weitaus schwieriger heraus. Bei SaaS wird im Regelfall bereits verwendbare Software bereitstellt, welche für einen speziellen Zweck entworfen wurde. Da es sich dabei um die unterschiedlichsten Ausprägungen von Software handelt, ist eine sinnvolle Standardisierung kaum möglich (vgl. [2]). In der PaaS-Kategorie hingegen besteht das Problem, dass in diesem Bereich normalerweise ein großes Paket aus verschiedenen Laufzeitumgebungen, Middleware, Frameworks und Services bereitgestellt wird, welches sich von Anbieter zu Anbieter stark unterscheiden kann. Durch den Mangel an Standardisierung kommt es bei Nutzern von PaaS-Systemen häufig zu einem sogenannten *Vendor-Lock-In*. Das bedeutet, dass es Nutzern eines PaaS-Angebotes schwer gemacht wird, zu einem anderen Anbieter zu wechseln. Meist ist dies nur in Verbindung mit einem erheblichen Organisations- und Konfigurationsaufwand möglich, wodurch dem Anwender zusätzliche Kosten entstehen (vgl.[2]). Neben der fehlenden Standardisierung technischer Aspekte der PaaS-Systeme kommt noch hinzu, dass im Vergleich zu IaaS und SaaS kaum einheitliche Preisstrukturen existieren. Bei SaaS-Produkten, welche meist ein vordefiniertes Gesamtpaket bereitstellen, wird bevorzugt auf feste Preisstrukturen gesetzt. Dazu zählen Abonnements und nutzungsbasierte *Pay-as-you-go*-Modelle. IaaS-Lösungen hingegen sind meist sehr flexibel gestaltet und der Nutzer kann die Konfiguration seines Systems exakt an seine Anforderungen anpassen. Aus diesem Grund kommen hier oftmals Preismodelle zum Einsatz, welche Preise dynamisch berechnen. Dabei orientiert sich die Höhe des Preises an den Kennzahlen des Systems, wie beispielsweise der Anzahl der CPU-Instanzen, des zur Verfügung gestellten Speichervolumens oder der maximalen Bandbreite der Netzwerkverbindung(vgl. [3]).

Im Gegensatz dazu hat sich im PaaS-Bereich bisher keine einheitliche Preisstruktur durchsetzen können. Da PaaS zwischen IaaS und SaaS angesiedelt ist und einen großen Bereich an Systemen abdeckt, sind die Ausprägungen von PaaS-Systemen sehr breit gefächert. So unterscheiden sich Systeme dieser Kategorie oftmals sehr in ihren Abstraktionsgra-

den. Einige PaaS-System orientieren sich stark an IaaS und bieten dem Nutzer durch den niedrigeren Abstraktionsgrad eine höhere Flexibilität. Wiederum andere schlagen mehr die Richtung von SaaS-Systemen ein, abstrahieren also mehr von der Hardware, Betriebssystem und Middleware und bieten dem Nutzer schon integrierte Softwarelösungen. Ein PaaS kann zwischen diesen beiden Extremen jeden Abstraktionsgrad annehmen. Dieses breite Spektrum an PaaS-Systemen hat zur Folge, dass hier viele verschiedene Preismodelle zum Einsatz kommen. So finden sowohl fixe Modelle, wie im SaaS-Bereich üblich, Anwendung, als auch die in der SaaS-Kategorie geläufigen dynamischen Varianten. Es ist somit für einen Nutzer nicht immer klar zu erkennen, welche Kosten für ein PaaS-System, welches den Anforderungen des Nutzers entspricht, anfallen. Da die Preisberechnung bei einzelnen Anbietern von unterschiedlichen Faktoren abhängt, kann ein und dasselbe System von Anbieter zu Anbieter unterschiedliche Kosten verursachen. Um eine bessere Vergleichbarkeit zwischen Preismodellen von PaaS-Angeboten zu gewährleisten, wird hier eine Klassifikation der Parameter, welche Einfluss auf die Preisberechnung haben, vorgenommen werden. Auf Grundlage dieser Klassifikation wird anschließend ein einheitliches Datenmodell entworfen werden, welches die verschiedenen Parameter abbilden kann, um damit eine Vergleichbarkeit zwischen Angeboten zu ermöglichen.

2 Problemstellung

Diese Bachelorarbeit setzt sich zum Ziel, eine Klassifikation von Preismodellen verschiedener PaaS-Anbieter vorzunehmen, um eine leichtere Vergleichbarkeit zwischen den Modellen herzustellen. Um dieses Ziel zu erreichen, wird als erster Schritt eine Datenerhebung anhand der aktuell auf *PaaSfinder.org* vertretenen Anbieter durchgeführt. Dadurch sollen Erkenntnisse darüber gewonnen werden, wie sich die Preismodelle der einzelnen Anbieter zusammensetzen und durch welche Eigenschaften der Preis beeinflusst werden kann. Die daraus gewonnenen Informationen sollen anschließend dazu beitragen, ein Datenmodell zu entwerfen, welches in der Lage ist, die Preismodelle aller Anbieter vollständig und einheitlich abzubilden, so dass eine Gegenüberstellung möglich wird. Mit Hilfe dieses Datenmodells soll für jeden Anbieter ein Profil erstellt werden, in dem dessen Tarife abgebildet sind. Für jeden Tarif soll dabei genau spezifiziert werden, welche Parameter den Preis des Tarifs auf welche Art beeinflussen. Beispielsweise gibt es Tarife, bei denen präzise einzelne Optionen, wie die Anzahl der CPU-Instanzen und die Größe des RAMs ausgewählt werden können. Andere Tarife stellen eher feste Gesamtpakete dar, zwischen denen der Anwender wählen kann. Im ersten Fall ist der Preis sehr flexibel und im Detail von der genauen Konfiguration abhängig, während im zweiten Fall der Preis eher statisch ist. Das Datenmodell soll in der Lage sein beides abzudecken.

Ein weiteres, wichtiges Kriterium beim Entwurf des Modells stellt die Flexibilität, Wartbarkeit und Erweiterbarkeit dar. Gerade aufgrund des breiten Spektrums an PaaS-Angeboten ist es wichtig, dass jederzeit die Möglichkeit besteht, neu hinzukommende Anbieter zu integrieren oder sich ändernde Angebote zu übernehmen, ohne dabei Änderungen am bestehenden Modell vornehmen zu müssen. Daher ist es für den Entwurf des Datenmodells wichtig, vom eigentlichen Inhalt der Tarife zu abstrahieren und stattdessen zu evaluieren, in welcher Form ein Preis in einem Tarif beeinflusst werden kann. Dadurch soll ermöglicht werden, dass auch ein neu hinzukommender Tarif, der auf Parameter setzt, welche bisher noch nicht vorkommen, problemlos integriert werden kann.

Dieses Datenmodell soll anschließend als Grundlage einer Webapplikation dienen, mit deren Hilfe Anwender, die auf der Suche nach einem geeigneten PaaS-Anbieter sind, die Möglichkeit bekommen sollen, die Preise verschiedener Anbieter miteinander zu vergleichen. Dabei soll der Nutzer in der Lage sein, über ein Webinterface die Eigenschaften, die das gewünschte System aufweisen soll, nach Belieben auszuwählen und zu konfigurieren. Beispielsweise soll ein Nutzer hardwarespezifische Eigenschaften, wie die Anzahl der CPU-Instanzen und die Größe von Arbeits- und Hintergrundspeicher, sowie softwarespezifische Eigenschaften, wie unterstützte Frameworks, Middleware und Laufzeitumgebungen wählen können. Daraufhin werden die vorhandenen Anbieterprofile nach passenden Angeboten durchsucht und der jeweilige Preis berechnet. Dem Nutzer sollen anschließend die Ergebnisse, dem Preis nach aufsteigend geordnet, präsentiert werden. Dabei soll direkt ersichtlich sein, bei welchem Anbieter mit welchen Kosten zu rechnen ist. Zudem ist es wichtig, dass die exakte Systemkonfiguration angezeigt wird, die der entsprechende Tarif bietet, da dort oftmals auch Eigenschaften enthalten sind, welche der Nutzer nicht explizit ausgewählt hat.

3 Verwandte Arbeiten

Die steigende Popularität und kommerzielle Verbreitung von Cloud-Plattformen hat auch dazu geführt, dass sich die Wissenschaft zunehmend mit der Preisgestaltung von Cloud-Angeboten beschäftigt. Dabei befasste man sich unter anderem mit der Klassifizierung verschiedener Preismodelle und welche Vor- und Nachteile diese bieten. Zudem wurden Berechnungsmethoden vorgestellt, welche es dem Anbieter ermöglichen sollen, den optimalen Preis für sein Angebot zu bestimmen. Des Weiteren wurde bereits erarbeitet, welche Probleme bei der Standardisierung von PaaS auftreten können.

Al-Roomi *et al.* (vgl. [4]) stellten sich die Frage, welche Preismodelle für Cloudplattformen existieren und welche Auswirkungen diese auf Anbieter und Nutzer haben. Demnach können Preismodelle in die drei Kategorien *fixed*, *dynamic* und *market-dependent* unterteilt werden.
In den Bereich *fixed* fallen dabei Abomodelle und das sogenannte *pay-per-use*-, beziehungsweise *pay-as-you-go*-Modell. Bei einem Abomodell zahlt der Anwender einen festen Betrag im Voraus und erhält im Gegenzug einen genau definierten Service vom Anbieter über einen festgelegten Zeitraum. Das *pay-as-you-go*-Modell hingegen berechnet dem Nutzer einen festen Preis pro genutzter Einheit. Im Regelfall ist damit die Nutzung pro Zeiteinheit gemeint. Beispielsweise berechnet *Amazon* einen festen Betrag pro Stunde, die eine Virtuelle Maschine genutzt wird (vgl. [5]). Fixe Preismodelle finden besonders häufig bei der Berechnung von SaaS-Systemen Verwendung, da diese zumeist einen sehr speziellen Service anbieten und daher keine komplexe Preisberechnung notwendig ist (vgl. [3]).
Preismodelle, die unter die Kategorie *dynamic* (auch *metered* genannt) fallen, berechnen den Preis dynamisch in Abhängigkeit von der Beschaffenheit des Systems und der Nutzungsmenge des Anwenders. Dabei haben vor allem hardwarespezifische Eigenschaften des Systems, wie die Rechenleistung der CPUs, die verwendete Speichermenge und das Volumen des Netzwerk-Traffics, Einfluss auf den Preis. Die dem Nutzer entstehenden Kosten berechnen sich in diesem Fall daraus, wieviel einer Ressource der Nutzer in einem bestimmten Zeitraum in Anspruch nimmt (vgl. [4], [3]). Preismodelle dieser Art sind

prädestiniert für Cloud-Angebote aus dem IaaS-Bereich. Das hat den Grund, dass für eine einheitliche Messbarkeit von Ressourcen eine gewisse Standardisierung von Nöten ist, was bei IaaS der Fall ist (vgl. [3]). Oftmals werden im IaaS-Bereich fixe und dynamische Preismodelle kombiniert. Dies erfolgt zumeist durch einen Festbetrag bis zu einer gewissen Grenze und dynamischen Zusatzkosten, sobald diese Grenze überschritten wird (vgl. [3]). Bei *Market-dependent* Preismodellen richten sich die Kosten eines Cloud-Systems ausschließlich nach Angebot und Nachfrage. Darunter fallen unter anderem Auktionsmodelle, bei denen Cloud-Services demjenigen verkauft werden, der am meisten bietet. Eine weitere Möglichkeit, in dieser Kategorie einen Preis festzulegen, besteht in einer direkten Verhandlung zwischen Anbieter und Nutzer (vgl. [3]).

Des Weiteren hat man sich in [2] bereits ausführlicher mit dem Thema PaaS und den damit einhergehenden Portabilitätsproblemen befasst. So wurde aufgrund der Tatsache, dass der Begriff PaaS ein so breites Spektrum abdeckt, eine weitere Unterteilung in Unterkategorien vorgeschlagen, abhängig vom Abstraktionsgrad. Je nachdem, ob ein PaaS-System sich eher an IaaS oder SaaS orientiert, kann das System als *IaaS-centric* oder *SaaS-centric PaaS* klassifiziert werden. Ein System, welches sich zwischen diesen beiden Extremen befindet, lässt sich als *Generic-PaaS* einordnen. Gerade im Bezug auf verschiedene Preismodelle, welche abhängig von SaaS oder IaaS unterschiedlich ausfallen, stellt diese Einteilung eine Möglichkeit dar, die Preismodelle von PaaS-Systemen vergleichbarer zu gestalten.

Um die Standardisierung und Portabilität von PaaS weiter voranzutreiben, wurde zudem die Plattform *PaaSfinder.org* geschaffen (vgl. [2]). Dahinter verbirgt sich eine Webanwendung, die den direkten Vergleich von PaaS-Angeboten ermöglicht. Sie basiert auf einer Sammlung einheitlich definierter Anbieterprofile, welche alle Informationen über diesen Anbieter enthalten. Dies beinhaltet sowohl technische Details wie beispielsweise die unterstützten Laufzeitumgebungen und Middleware, als auch geschäftsbezogene Informationen über das Preismodell und die Compliancestandards. Anhand dieser Anbieterprofile lassen sich die Eigenschaften einzelner Anbieter gegenüberstellen und vergleichen.

4 Datenerhebung und Modellentwurf

Der erste Teil dieser Bachelorarbeit befasst sich mit dem Entwurf eines geeigneten Datenmodells, das in der Lage sein soll, die unterschiedlichen Preisstrukturen aktueller PaaS-Angebote in eine einheitliche Form zu bringen, um eine Vergleichbarkeit zwischen verschiedenen Angeboten zu ermöglichen. Dieses Datenmodell soll dazu verwendet werden, die Preismodelle der auf *PaaSfinder.org* gelisteten Cloud-Anbieter so vollständig wie möglich abzubilden.

Um ein besseres Bild davon zu bekommen, welche Wege einzelne Anbieter bei der Gestaltung ihrer Preismodelle einschlagen, wurde als erstes eine Datenerhebung durchgeführt. Dazu wurden Informationen über die Anbieter, welche aktuell auf *PaaSfinder.org* vertreten sind, gesammelt und analysiert. Es wurden Details, wie Preismodelle, Zahlungsintervalle, Währungen und Parameter, welche den Preis beeinflussen können, untersucht. Daraus sollten Informationen gewonnen werden, wie breit gefächert die einzelnen Angebote sind und wie flexibel dementsprechend das Datenmodell sein muss. Basierend auf den gesammelten Informationen wurde das Datenmodell entworfen. Eine hohe Priorität besaß dabei das Ziel, das Modell möglichst flexibel zu gestalten. Es sollte verhindert werden,

dass zukünftige Änderungen von Anbietern an ihren Preisprofilen nicht vom Modell abgebildet werden können und dadurch grundlegende Anpassungen am System nötig werden. Auch neue Anbieter sollten problemlos integriert werden können, unabhängig von welchen Parametern deren Preismodelle abhängen.

Deshalb wurde beim Entwurf weniger darauf eingegangen, welche Parameter im einzelnen für einzelne Anbieter relevant sind. Stattdessen sollte erarbeitet werden, auf welche Art und Weise ein einzelner Parameter Einfluss auf den Preis eines Cloud-Systems haben kann. Es sollte ein Klassifikationssystem entstehen, welches es ermöglicht, jede Art von Parameter exakt einer der Klassen zuzuweisen und dementsprechend zu verarbeiten.

4.1 Datenerhebung

Für die Datenerhebung, welche die Grundlage für die Erstellung des Datenmodells darstellt, wurden als Stichprobe alle vorhandenen Cloudanbieter auf *PaaSfinder.org* herangezogen, da diese eine große Bandbreite unterschiedlicher PaaS-Angebote abdecken. Da der *Platform-as-a-Service*-Bereich des Cloud-Computings sehr breit gefächert ist, sind hier dementsprechend sowohl Anbieter der *IaaS-Centric*-, *Generic*-, als auch *SaaS-Centric*-Kategorie vertreten. Das bietet zwar zum einen ein sehr heterogenes Feld an Anbieterinformationen, birgt allerdings auch die Gefahr, dass Angebote auch einfach zu grundverschieden sein können, was eine Vergleichbarkeit erschwert, in Einzelfällen sogar unmöglich macht.

Zum Zeitpunkt der Durchführung der letzten Iteration der Datensammlung am 28.04.2017 befanden sich 71 verschiedene Anbieter in der Datenbank der Anwendung. Die bisherigen Anbieterprofile enthielten zwar schon Informationen über die verwendeten Preismodelle, allerdings handelte es sich dabei lediglich um die Informationen, ob der Anbieter auf ein *fixed*, ein *metered* oder ein kostenfreies Modell setzt.
Bei der Datenerhebung wurde auch überprüft, ob diese Daten jeweils noch aktuell sind. Ergänzend zu diesen Informationen wurden Daten über drei weitere Faktoren gesammelt. Genauer gesagt ist es für die Berechnung des Preises relevant zu wissen, welche Kenngrößen die Preise beeinflussen und auf welche Art sie dies tun. Des Weiteren ist die Tatsache interessant, ob die Anbieter eher auf ein *fixed*-, *dynamic*- oder *market-dependant*-Preismodell setzen. Dabei kann zudem überprüft werden, ob die Verteilung der Preismodelle bei *IaaS-centric* und *SaaS-centric* PaaS sich ähnlich verhält wie bei IaaS und SaaS. Wäre das der Fall, würde das bedeuten, dass Anbieter von *IaaS-centric* PaaS tendenziell eher auf dynamische Preismodelle setzen, während Angebote aus dem *SaaS-centric* PaaS-Bereich eher ein fixes Preismodell aufweisen würden.

4.1.1 Bestimmung der Preisparameter

Zu bestimmen, von welchen Parametern genau einzelne Anbieter ihre Preise abhängig machen, ist für den Entwurf des Datenmodells im Grunde nicht von Bedeutung. Der Grund dafür ist, dass bei der Klassifikation der Kenngrößen lediglich die Art und Weise, wie diese den Preis beeinflussen, von Relevanz ist, nicht aber, um welchen Wert es sich genau handelt.

Allerdings liefern die verwendeten Parameter eine Übersicht darüber, wie breit gefächert

diese im Großen und Ganzen sind und wie stark sich dementsprechend die Modelle der Anbieter unterscheiden können. Die Analyse aller 71 Anbieterprofile hat unter anderem eine sehr breite Parameterverteilung ergeben. Insgesamt wurden 69 unterschiedliche Kenngrößen identifiziert, welche in unterschiedlicher Form Teile von Anbietertarifen sind. Dies würde bedeuten, dass annähernd jeder Anbieter im Schnitt einen neuen Parameter einführt. Hinzu kommt, dass einige Anbieter Technologien in ihre Tarife mit einbeziehen, die komplett anbieterspezifisch sind. Beispielsweise bietet das Softwareunternehmen *SAP* für ihre Cloud-Plattform *HANA* als Teil ihrer Tarife Services wie die *SAP Cloud Platform IDE* oder das *SAP Cloud Platform Portal* an. Für Dienste dieser Art gibt es oftmals keine entsprechenden Pendants anderer Anbieter und wenn doch, ist es schwer zu bestimmen, in wie weit sich diese vergleichen lassen. Aus diesem Grund wurden Parameter, die einen speziellen Service des Anbieters repräsentieren, nicht mit in die Datensammlung aufgenommen.

In die Gesamtzahl der 71 Anbieter fließen allerdings auch Anbieter mit ein, welche ihre Services kostenlos zur Verfügung stellen, und dementsprechend auch über keine Preisprofile mit entsprechenden Parametern verfügen. Zum Zeitpunkt der Studie gehörten neun Anbieter zu dieser kostenfreien Kategorie.
Hinzu kommt, dass die Datenerhebung stark von der Transparenz der Anbieter abhängt. Einige Provider, welche kostenpflichtige Cloud-Services zur Verfügung stellen, machen keinerlei Informationen darüber öffentlich, welche Kosten dafür für den Nutzer anfallen würden. Zumeist wird in diesen Fällen lediglich die Möglichkeit geboten, Kontakt zum Anbieter aufzunehmen, um direkt abklären zu können, welche Anforderungen der Nutzer an das gewünschte System besitzt und dementsprechend wohl auch, auf welchen Betrag sich die Kosten belaufen würden. Das mag zwar für interessierte Kunden einen praktischen Service darstellen, trägt aber keineswegs zu einer besseren Preistransparenz unter Anbietern bei und schränkt damit die Vergleichbarkeit stark ein.
Provider, über die keine Preis- und Tarifinformationen gefunden werden konnten, waren auf *PaaSfinder.org* überraschend häufig vertreten. 21 Anbieter, und damit über 25% der Gesamtzahl, stellen auf ihren Webauftritten keinerlei Preisinformationen bereit. Zusätzlich dazu waren zum Zeitpunkt der ersten Iteration der Datenerhebung einige Anbieter auf *PaaSfinder.org* vertreten, deren Webseiten entweder nicht erreichbar waren, oder aber dort klar zu erkennen gaben, dass sie den Betrieb ihrer Cloud-Angebote eingestellt hatten. Bei der letzten Überprüfung der Anbieterprofile waren die vier betroffenen Anbieter allerdings bereits nicht mehr auf *PaaSfinder.org* vertreten.

Von den 71 ursprünglichen Anbietern ließen sich also, nach Abzug aller kostenlosen, zurückgezogenen oder intransparenten Anbieter, gerade einmal aus 37 Informationen über verwendete Parameter ziehen. Aufgrund der kleinen Menge ist es umso überraschender, dass die Anzahl unterschiedlicher Parameter mit 69 doch so hoch ausfällt. Die Zahlen zeigen deutlich, wie unterschiedlich die Preisgestaltungen einzelner Anbieter im PaaS-Bereich ausfallen. Dies hat seinen Hauptgrund in der breiten Beschaffenheit des PaaS-Bereichs. Wie bereits erwähnt, lässt sich der PaaS-Bereich in die drei Unterkategorien *IaaS-Centric*-, *Generic*-, und *SaaS-Centric* unterteilen, je nachdem wie nah sich das jeweilige System am *IaaS*-, beziehungsweise am *SaaS*-Bereich befindet. Dementsprechend fallen unter den Begriff *Plattform-as-a-Service* viele verschiedene Anbieter, welche vollkommen unterschiedliche Dienste anbieten. Beispielsweise lassen sich beim Anbieter *Microsoft* mit seiner Cloud-Plattform *Azure*, welche der Kategorie *IaaS-Centric* angehört, detaillierte Hardwarekonfigurationen zusammenstellen. Diese steht dem Nutzer zur freien Verfügung, wie im *IaaS*-Bereich üblich. Dem gegenüber stehen Anbieter der *SaaS*-Kategorie, wie

zum Beispiel *Mendix*, wo vollkommen von der Hardware abstrahiert wird und stattdessen Kenngrößen wie *Scaling* und *Failover* Teil der Tarife sind.

Da schon allein aus Gründen der Usability nicht jeder einzelne der 69 Parameter im Preisvergleich berücksichtigt werden soll, wurde anschließend eine Auswahl getroffen. Es wäre zwar theoretisch möglich, die vollständige Liste an Parametern zu verwenden, allerdings würde die Benutzbarkeit der Preisvergleichs-Webseite darunter leiden. 35 der gefundenen Kenngrößen, und damit fast 50%, kommen jeweils lediglich bei einem einzigen Anbieter vor. Auf eine Preisvergleichs-Software übertragen bedeutet dies, dass die Anwendung mit Auswahlmöglichkeiten überladen wäre. Die Auswahl eines dieser Werte hätte dabei zur Folge, dass dem Nutzer nur noch ein einziger Provider angeboten werden würde. Eine wirkliche Vergleichbarkeit wäre dadurch nicht mehr gegeben. Auf der anderen Seite steigt mit der Anzahl der angebotenen Parametern auch die Vollständigkeit des Vergleichsportals. Je mehr Parameter dem Nutzer zur Auswahl stehen, desto exakter können die Ergebnisse an dessen Bedürfnisse angepasst werden.
Bei der Auswahl der geeigneten Parameter muss daher eine Menge gefunden werden, welche einen Mittelweg zwischen Vollständigkeit und Benutzbarkeit ermöglicht. Die Entscheidung, welcher Parameter integriert wird und welcher nicht, wird schlicht und ergreifend anhand der ermittelten Häufigkeit getroffen.

Wie aus *Tabelle 1* (vgl. Anhang) zu entnehmen ist, ist sind zwei Drittel aller vorkommenden Parameter zweimal oder seltener für den Tarif eines Anbieters relevant. Für diese Arbeit wird die Anzahl von mindestens vier als Schwellenwert herangezogen, um darüber zu entscheiden, ob der Parameter integriert werden soll oder nicht. Bei der aktuellen Zusammensetzung der Daten ergibt sich daraus ein Modell mit 16 verschiedenen Parametern.

4.1.2 Klassifikation von Parametern

Ein weiterer Aspekt, der mit Hilfe der Datenerhebung erarbeitet wurde, war es, eine geeignete Klassifikiation von Kenngrößen zu erreichen. Zunächst einmal wird definiert, was in diesem Kontext mit Parameter gemeint ist.

Als Parameter wird von hier an eine Eigenschaft eines Cloud-Systems bezeichnet, welche Teil eines Tarifs eines Anbieters ist und somit Einfluss auf den Preis des Systems hat. Im Bezug auf einen Preisvergleich für Cloud-Plattformen stellen diese Parameter letztendlich die wählbaren Optionen dar, aus denen der Nutzer die gewünschte Konfiguration seines Cloud-Systems zusammenstellen kann.

Jeder Parameter verfügt dabei über eine Reihe von Eigenschaften, welche dessen Beschaffenheit genauer beschreiben. Im Folgenden werden die Eigenschaften, welche ein Parameter besitzen kann, aufgezählt.

Einheit: Ein Parameter kann eine Einheit besitzen. So werden beispielsweise Angaben zum Speicherplatz in der Regel in Gigabyte angegeben. Es ist aber auch möglich, dass ein Parameter nur über die reine Anzahl bestimmt wird und damit keine Einheit besitzt. Dies wäre zum Beispiel bei der Anzahl der CPU-Instanzen der Fall.

Einheitengröße: Oftmals werden Parameter in einem Tarif nicht in jeder erdenklichen Anzahl angeboten, sondern nur in bestimmten Intervallen. So kann beispielsweise im Normalfall der Speicherplatz für eine Cloudplattform nicht vollkommen frei gewählt werden,

sondern nur in festgelegten Intervallen von zum Beispiel 10GB.

Preis pro Einheit: Die Tatsache, dass die Menge nicht immer frei gewählt werden kann, sondern oftmals nur in bestimmten Einheitengrößen, hat zur Folge, dass auch der Preis von dieser Einheitengröße abhängt.

Zudem wurde, unabhängig von den Parametern selbst, untersucht, wie sich ein Parameter auf den Preis eines Cloud-Angebots auswirken kann. Dabei wurden zusätzliche Eigenschaften festgestellt, welche ein Parameter aufweist. Durch die Kombination dieser Eigenschaften ist es möglich, eine genaue Klassifizierung vorzunehmen.

Unabhängigkeit: Die erste dieser Eigenschaft, die ein Preisparameter in einem Tarif besitzen kann, ist *Unabhängigkeit*. Sie beschreibt, ob der Parameter einzeln auftritt oder immer nur in Preiskombination mit einem oder mehreren anderen. Ein unabhängiger Parameter kann in einem Tarif einzeln konfiguriert werden. Wenn beispielsweise bei einem Tarif frei ausgewählt werden kann, über wieviel Speicherplatz das gewünschte System verfügen soll, und dies keinerlei Auswirkung auf die restlichen Parameter hat, handelt es sich um einen unabhängigen Parameter. Für die Preisberechnung bedeutet dies, dass die Kosten für diesen Parameter schlicht zum Gesamtpreis hinzuaddiert werden können.
Wenn im Gegensatz dazu zum Beispiel ausgewählt werden kann, wie viele CPU-Instanzen das System besitzen soll, eine steigende Zahl der Instanzen aber automatisch mit einem steigenden RAM-Wert einhergeht, handelt es sich um abhängige Parameter, welche nur in einem *Bundle* verfügbar sind. Bei der Berechnung des Preises muss hier darauf geachtet werden, das die Auswahl eines abhängigen Parameters immer automatisch die Auswahl seiner Abhängigkeiten zur Folge hat, welche nicht zusätzlich berechnet werden.
Es ist zu beachten, dass ein bestimmter Parameter, beispielsweise Speicherplatz, in einem Tarif als unabhängig gelten kann, in einem anderen, möglicherweise sogar vom gleichen Anbieter, jedoch mit Abhängigkeiten versehen sein kann.

Skalierbarkeit: Ein weiteres Attribut, welches ein Parameter besitzen kann, ist die *Skalierbarkeit*. Diese Eigenschaft sagt aus, in welchen Ausprägungen ein Parameter auftreten kann. Dabei wird grundsätzlich zwischen *binären* und *nicht-binären* Parametern unterschieden. Ist ein Parameter als binär gekennzeichnet, besitzt also die Einheit *binär*, bedeutet dies, dass es nur zwei mögliche Optionen gibt. Der Parameter kann entweder Teil eines Tarifs sein oder nicht, eine Mengenangabe ist nicht vorhanden. Ein Beispiel für einen binären Parameter wäre eine *Backup-Funktion*, welche einer Systemkonfiguration hinzugefügt werden kann. Dem gegenüber stehen nicht-binäre Parameter, bei welchen sich exakt angeben lässt, wie viele Einheiten davon vorhanden, beziehungsweise gewünscht sind. Parameter dieser Art wären beispielsweise die Anzahl der CPU-Instanzen. Der Preis richtet sich hierbei nach der gewählten Anzahl. Im Gegensatz zur Unabhängigkeit gilt die Skalierbarkeit eines bestimmten Parameters über alle Anbieter hinweg. Ein Parameter kann also nicht bei einem Anbieter als binär klassifiziert sein, bei einem anderen jedoch als nicht-binär. Beispielsweise ist der Parameter CPU-Instanzen in jedem Fall ein nicht-binärer.

Linearität: Nicht-binäre Parameter besitzen noch eine zusätzliche Eigenschaft, die *Linearität*. Da bei einem nicht-binären Parameter mehrere Abstufungen als gewählte Menge möglich sind, steht der Preis immer in Relation zur Menge. Im einfachsten Fall steigt der Preis immer in gleichem Maße mit der Anzahl. In diesem Fall ist der Parameter linear. Oftmals kommt es jedoch vor, dass Preise sich mit steigender Menge verändern, beispielsweise durch einen Mengenrabatt. So kann zum Beispiel beim Parameter CPU-Instanzen im Intervall von zwei bis acht ein höherer Preis pro Einheit anfallen, als wenn mehr als

acht gewählt werden. Trifft dies auf einen Parameter zu, gilt er als nicht-linear. Wie auch bei der Unabhängigkeit ist es möglich, dass ein Parameter innerhalb eines Tarifs linear sein kann, innerhalb eines anderen kann er jedoch auch als nicht-linear auftreten.
Anhand dieser Eigenschaften ergeben sich nun sechs verschiedene Klassifizierungen von Parametern.

1. Unabhängiger, binärer Parameter:
Dies stellt die einfachste Form eines Parameters dar. Der Nutzer kann in diesem Fall schlicht auswählen, ob der Parameter Teil des Systems sein soll oder nicht. Ein Beispiel für einen unabhängigen, binären Parameter ist eine Monitoring-Funktion, welche vom Anwender zusätzlich zum System hinzugebucht werden kann. Da diese Funktion keine Abstufungen besitzt und entweder vorhanden ist oder nicht, gilt sie als binär. Da ein Hinzufügen der Funktion einzeln möglich ist, handelt es sich um einen unabhängigen Parameter.

2. Unabhängiger, nicht-binärer, linearer Parameter:
In diesem Fall kann der Parameter, genauso wie im Fall davor, einzeln konfiguriert werden. Dabei wird aber nicht nur ausgewählt, ob dieser Parameter vorhanden sein soll, sondern ebenso, wie viele Einheiten davon gewünscht sind. Mit jeder zusätzlichen Einheit wächst der Preis um den gleichen Wert. Ein Beispiel für einen Parameter dieser Art ist die Hintergrundspeichermenge eines Systems, die der Nutzer in Abstufungen auswählen kann und deren Preis pro Einheit sich nicht verändert.

3. Unabhängiger, nicht-binärer, nicht-linearer Parameter:
Ebenso wie im vorherigen Fall können für einen Parameter dieser Art unterschiedliche Mengenangaben gemacht werden. Der Unterschied hierbei ist jedoch, dass sich der Preis pro Einheit jeweils an bestimmten Schwellenwerten verändert. Als Beispiel dient auch hier wieder die vom Nutzer frei wählbare Speichermenge. Im Gegensatz zum vorherigen Fall, wird dem Anwender allerdings ab einer gewissen Menge ein Mengenrabatt pro Einheit gewährt.

4. Abhängiger, binärer Parameter:
Ein abhängiger, binärer Parameter, zeichnet sich dadurch aus, dass er Teil eines Bundles ist. Das bedeutet, dass er immer nur in Kombination mit anderen Parametern auftritt. Sobald einer dieser Werte ausgewählt wird, werden automatisch alle von diesem abhängigen Parameter zusätzlich ausgewählt. Dabei spielt es keine Rolle, ob die abhängigen Parameter binär oder nicht-binär sind. Beispielsweise kann es vorkommen, dass ein Anbieter eine Monitoringfunktion nur in Verbindung mit einer Loggingfunktion anbietet.

5. Abhängiger, nicht-binärer, linearer Parameter:
Dieser Fall beschreibt einen Parameter, welcher eine Mengenangabe besitzt, also nicht-binär ist und von anderen abhängig ist. Der Preis steigt dabei mit immer dem gleichen Wert pro Einheit. Sind zwei nicht-binäre, lineare Parameter voneinander abhängig, müssen diese, zur korrekten Berechnung, den gleichen Preis pro Einheit besitzen, die Größen der Einheiten selbst können jedoch unterschiedlich sein. Angenommen die Zahl der CPU-Instanzen und die Größe des RAMs wären voneinander abhängig, können beispielsweise an jede einzelne Instanz 256MB RAM geknüpft sein. Würde der Nutzer nun vier Instanzen auswählen, würde er das gleiche Angebot bekommen wie bei der Auswahl von 1024 MB RAM, nämlich eine Kombination aus beidem. Für jedes zusätzliche Paar aus einer Instanz und 256MB RAM würde dabei der Preis um denselben Wert (Preis pro Einheit) erhöht werden.

6. Abhängiger, nicht-binärer, nicht-linearer Parameter:
Verhält sich wie im Fall zuvor, jedoch mit dem Unterschied, dass sich der Preis ab einem bestimmten Schwellenwert verändert. Dabei ist zu beachten, dass die Intervalle, in welchen ein bestimmter Preis gilt, bei allen voneinander anhängigen Parametern gleich groß sein müssen. Gilt beispielsweise für bis zu vier CPU-Instanzen ein bestimmter Preis, ab der fünften aber ein günstigerer, muss ebenso für 256MB bis 1024MB RAM der Standardpreis, und ab allem darüber der für beide gültige günstigere Preis angewendet werden.

Diese Einteilung in unterschiedliche Klassen hat den Vorteil, dass vollständig von der Semantik eines Parameters abstrahiert werden kann. Das ermöglicht zu einem späteren Zeitpunkt auch neue Parameter hinzuzufügen, welche bisher noch überhaupt nicht vorgekommen sind. Solange ein Parameter einer der beschriebenen Klassen zugeordnet werden kann, lässt sich dieser auch abbilden.

4.1.3 Verteilung der Preismodelle

Im Zuge der Datenerhebung wurden außerdem Informationen darüber gesammelt, welche Arten von Preismodellen bei den Anbietern vertreten sind und wie diese sich auf die unterschiedlichen PaaS-Typen verteilen.
Von den aktuell 71 Anbietern auf *PaaSfinder.org* fallen 13 in die Kategorie *IaaS-centric* und elf in den Bereich *SaaS-centric*. Die restlichen 47 lassen sich der Kategorie *generic* zuordnen. Im *IaaS-centric*-Bereich befinden sich lediglich vier Anbieter, bei denen verwertbare Preisinformationen vorhanden sind. Dabei setzen jeweils zwei auf fixe und zwei auf dynamische Preismodelle.
Für *IaaS-centric* PaaS ergibt sich ein ähnliches Bild. Gerade einmal fünf der elf Anbieter dieser Kategorie bieten auf ihren Webauftritten Informationen über ihre Preisstrukturen an. Davon setzen drei auf ein fixes Preismodell, während die restlichen zwei dynamische Modelle verwenden.
Von den restlichen 47 *generic* PaaS ließen sich bei 28 Informationen über deren Preisgestaltung finden. Auch hier zeichnet sich in der Verteilung der Preismodelle ein sehr ausgeglichenes Bild ab. Dynamische Preismodelle sind dabei mit einer Anzahl von 15 etwas häufiger vertreten als fixe Modelle, welche nur bei 13 Anbietern Anwendung finden.
Es zeigt sich also, dass das Verhältnis unterschiedlicher Preismodelle im PaaS-Bereich ziemlich ausgeglichen ist. Dies gilt sowohl für den *IaaS-centric*-, den *SaaS-centric*- und den *generic*-Bereich. Es konnten keine Anhaltspunkte gefunden werden, dass *IaaS-centric* PaaS wie IaaS vermehrt auf dynamische Modelle setzt. Genausowenig lässt sich eine Aussage darüber treffen, ob Anbieter von *SaaS-centric* PaaS die im SaaS-Bereich präferierten fixen Modelle bevorzugen. Um eine endgültige Widerlegung oder Bestätigung dieser Theorie machen zu können, ist jedoch eine Untersuchung einer weitaus größeren Menge an Anbietern von Nöten.
Über die Verbreitung von Preismodellen, welche in die Kategorie *market-dependent* fallen, lässt sich nur schwer eine Aussage treffen. Das hat hauptsächlich den Grund, dass bei Modellen dieser Art die Preise nicht von vornherein festgelegt sind, sondern im Laufe eines Verhandlungs- oder Auktionsprozesses ermittelt werden. Aus diesem Grund verzichten Anbieter in diesen Fällen höchstwahrscheinlich darauf, ihre Preise transparent darzustellen. So liegt die Vermutung nahe, dass ein nicht unerheblicher Anteil der Anbieter, die keine Informationen bezüglich ihrer Preisstrukturen zu Verfügung stellen, auf marktabhängige Preismodelle setzen.

4.2 Der Modell-Entwurf

Nach dem Abschluss der Datenerhebung sollte ein Modell entworfen werden, welches die Möglichkeit bietet, die jeweiligen Tarife unterschiedlicher Anbieter korrekt und vollständig abzubilden. Dabei sollte es möglich sein, auf Grundlage dieses Modells für jeden Anbieter ein Profil zu erstellen, welches alle für einen Preisvergleich relevanten Daten enthält. Neben Daten, welche zur Preisberechnung benötigt werden, gehören dazu auch allgemeine Informationen über den Anbieter und dessen Tarife selbst. Die Profile sollen anschließend als JSON-Dokumente gespeichert werden, auf deren Grundlage der Preisvergleich zwischen Anbietern auf *PaaSfinder.org* aufgebaut werden soll.

4.2.1 Beschreibung des Modells

Die Informationen der Preisprofile können anhand ihrer Bezugspunkte in vier verschiedene Kategorien unterteilt werden. Je nachdem, ob diese sich auf den Anbieter selbst, auf einen Tarif, auf dessen Parameter oder auf einzelne Preise beziehen, lässt sich von *anbieter-*, *tarif-*, *parameter-* oder *preisspezifischen* Daten reden.

Zur vollständigen Modellierung eines Anbieterprofils wurden demnach die vier Datenobjekte *VendorPricing*, *Tarif*, *Parameter* und *Price* entworfen, welche voneinander abhängig sind und eine Baumstruktur bilden. *Abbildung 1* zeigt die vier Klassen und deren Relationen in der Übersicht.

VendorPricing: Das Wurzelelement der Baumstruktur stellt dabei das Objekt *VendorPricing* dar, welches allgemeine Informationen über den Anbieter selbst enthält. Dazu zählen der Name (*name*) des Anbieters, die verwendete Währung (*currency*), sowie ein Datum (*date*), welches aussagt, wann das Profil das letzte Mal aktualisiert wurde. Jede dieser drei Angaben ist zwingend erforderlich.

Tarif: Das zweite Objekt *Tarif* repräsentiert einen einzelnen Tarif. Da jeder Anbieter mehrere Tarife besitzen kann, jeder Tarif jedoch immer einem bestimmten Anbieter zugeordnet ist, besteht zwischen diesen beiden Objekten eine 1:n-Beziehung.
Dieses Objekt enthält die tarifspezifischen Daten, wie den Namen (*name*) des Tarifs und den Zeitraum (*interval*), in welchem die Kosten für den Tarif anfallen. Wie schon beim vorherigen Objekt muss jedes Attribut des Tarifs angegeben werden. Für das Zahlungsintervall gilt zudem noch die Einschränkung, dass dieses nur die Werte *daily*, *monthly* und *annually* annehmen kann, was der Menge der von den Anbietern verwendeten Zahlungsintervallen entspricht.

Parameter: Ein Tarif kann mehrere Parameter beinhalten, welche für diesen von Relevanz sind. Dies geschieht wiederum mittels einer 1:n-Beziehung zwischen einem *Tarif* und Datenobjekten vom Typ *Parameter*.
Ein *Parameter*-Objekt enthält dabei alle parameterspezifischen Informationen, wie den Namen (*name*) des Parameters, dessen Einheit (*unit*), sowie eine mögliche Maximalmenge (*upper_bound*). Außerdem wird in diesem Objekt angegeben, ob es sich um einen unabhängigen Parameter handelt oder ob dieser von anderen abhängig ist (*bundle*). Sollten zwei Parameter voneinander abhängig sein, wird dies dadurch gekennzeichnet, dass ihr *bundle*-Attribut jeweils dieselbe Nummer hat. Im Gegensatz zu Objekten der Typen *VendorPricing* und *Tarif* muss hier jedoch nicht jedes Attribut zwingend angegeben werden.

Lediglich *name* und *upper_bound* sind obligatorisch. Es kommt allerdings vor, dass ein Parameter keine Einheit besitzt, weswegen die Angaben des Attributs *unit* optional ist. Handelt es sich um einen binären Parameter, muss als Einheit der Wert *binary* angegeben werden und die Obergrenze muss den Wert 1 betragen. Das Weglassen der Eigenschaft *bundle* hingegen deutet auf einen unabhängigen Parameter hin.

Price: Ein Parameter kann mehrere Preise besitzen, um beispielsweise Fälle wie Mengenrabatte abbilden zu können. Aus diesem Grund besteht auch hier wieder eine 1:n Beziehung zwischen dem *Parameter*-Objekt und dem letzten Objekt der Hierarchie, dem *Price*.
Ein Preisobjekt enthält preisspezifische Informationen, wie die Einheitengröße (*pricing_unit*) und den Preis pro Einheit (*price_per_unit*). Für den Fall, dass sich die Preise des Parameters nicht linear verhalten, wird zusätzlich mittels einer Minimalmenge (*minimum*) und einer Maximalmenge (*maximum*) das Mengenintervall definiert, in welchem der entsprechende, feste Preis gilt. Für dieses Objekt gilt, dass nur die *pricing_unit*, sowie der *price_per_unit* zwingend erforderlich sind. Die Attribute *minimum*, und *maximum* werden hingegen bei binären Parameter weggelassen.

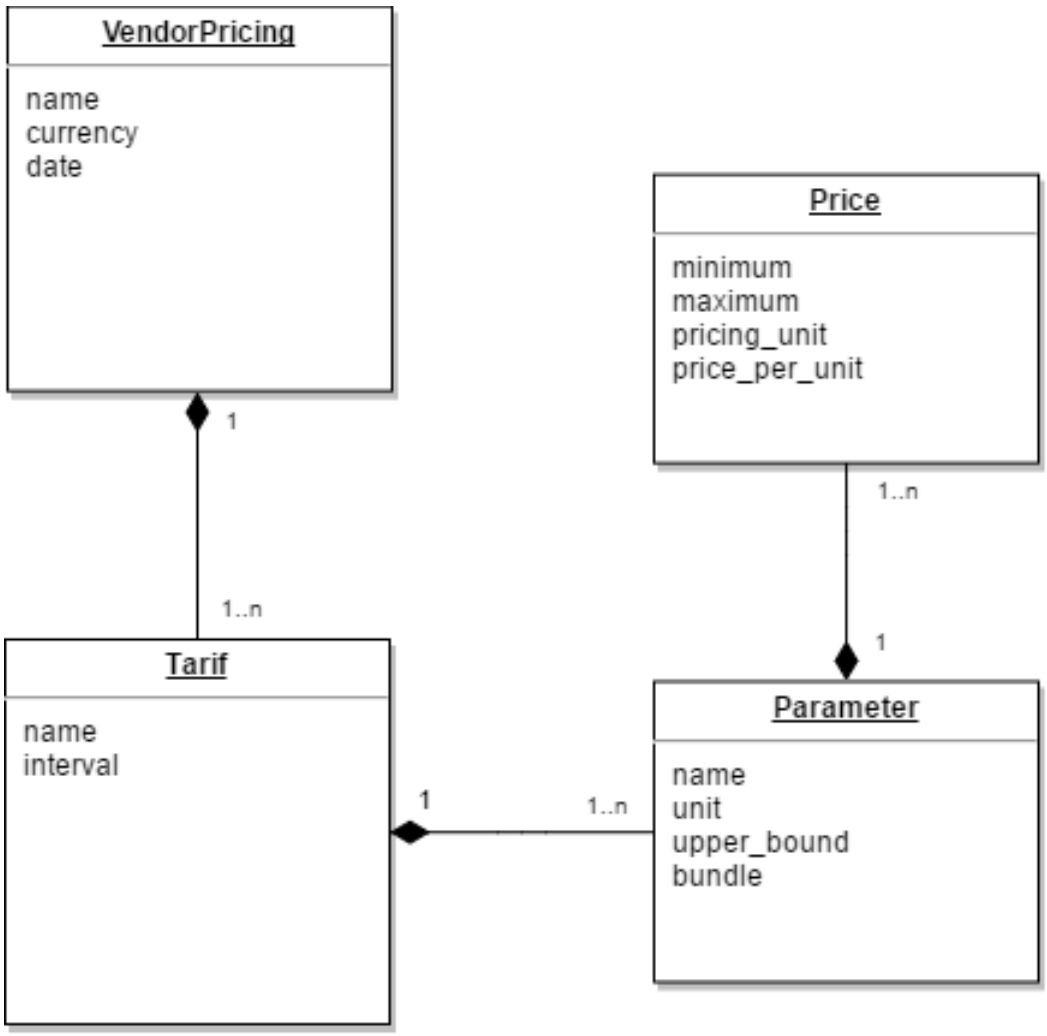

Abbildung 1: Datenmodell

4.2.2 Darstellung des Modells in JSON

Mit Hilfe der vier Datenobjekte lassen sich nun Anbieterprofile erstellen, welche alle nötigen Informationen für einen Preisvergleich enthalten. Durch die gegebene Baumstruktur ist es problemlos möglich, ein Anbieterprofil mit Hilfe eines JSON-Dokuments darzustellen. *Listing 1* zeigt beispielhaft ein Anbieterprofil mit jeweils einem Tarif, einem Parameter und einem Preis als Unterobjekte. Das Modell ist so konzipiert, dass es problemlos verschiedenste Arten von Preismodellen abbilden kann. Es wurde besonders darauf geachtet, dass es möglich ist, alle vorher auf Grundlage der Datenerhebung definierten sechs Klassifizierungen von Parametern abzubilden. Dadurch wird gewährleistet, dass das Modell flexibel genug ist, um mit zukünftigen Änderungen und Neuerungen umzugehen.

```
1  {
2    "name": "Provider1",
3    "date": "01.01.2017",
4    "currency": "EUR",
5    "tarif": [
6      {
7        "name": "Tarif1",
8        "interval": "monthly",
9        "parameter": [
10         {
11           "name": "Parameter1",
12           "upper_bound": 10,
13           "price": [
14             {
15               "minimum": 1,
16               "maximum": 10,
17               "pricing_unit": 1,
18               "price_per_unit": 100
19             }
20           ]
21         }
22       ]
23     }
24   ]
25 }
```

Listing 1: Anbieterprofil als JSON

Im Folgenden wird anhand von Beispielen beschrieben, auf welche Art sich die sechs Parameterklassen in einem Anbieterprofil darstellen lassen. Dabei werden zur Veranschaulichung real existierende Parameter verwendet.

1. Unabhängiger, binärer Parameter:
Ein Beispiel für einen unabhängigen, binären Parameter stellt eine Monitoringfunktion dar. *Listing 2* zeigt, in welcher Form ein solcher Parameter in einem Anbieterprofil dargestellt wird. Die Unabhängigkeit wird dabei modelliert, indem das Attribut *bundle* für den Parameter nicht gesetzt ist, da dieser kein Teil eines Bundles ist. Die Tatsache, dass es sich um einen binären Parameter handelt, wird hingegen durch mehrere Eigenschaften festgelegt. Zum einen wird dem Attribut *unit* des Parameters der Wert *binary* zugewiesen. Zusätzlich wird die Maximalmenge *upper_bound* auf eins begrenzt. Die Attribute *minimum* und *maximum* des hinzugegehörigen Preisobjekts sind hier nicht relevant und

können weggelassen werden.

```
{
  "name": "Provider1",
  "date": "01.01.2017",
  "currency": "EUR",
  "tarif": [
    {
      "name": "Tarif1",
      "interval": "monthly",
      "parameter": [
        {
          "name": "Monitoring",
          "unit": "binary",
          "upper_bound": 1,
          "price": [
            {
              "pricing_unit": 1,
              "price_per_unit": 20
            }
          ]
        }
      ]
    }
  ]
}
```

Listing 2: Unabhängiger binärer Parameter

2. Unabhängiger, nicht-binärer, linearer Parameter:
Ein unabhängiger, nicht-binärer, linearer Parameter, wie beispielsweise die Speichermenge, wird in *Listing 3* abgebildet. Im Gegensatz zum vorigen Fall trägt das Attribut *unit* hier nicht den Wert *binary*, sondern die zum Parameter passende Einheit *GB*. Zusätzlich dazu ist die Obergrenze für den Parameter (*upper_bound*) größer als eins, in diesem Fall liegt sie bei 100GB. Das hat außerdem zur Folge, dass die Eigenschaft *maximum* des Preises auf den gleichen Wert wie die Obergrenze gesetzt wird. Der Wert für das Attribut *minimum* weist dementsprechend einen beliebigen Wert auf, welcher kleiner oder gleich dem Maximum ist. Eine weitere Einschränkung für den Minimal- und Maximalwert ist, dass beide einen Wert aufweisen müssen, welcher ein ganzzahliges Vielfaches der Einheitengröße *pricing_unit* beträgt.

```
{
  "name": "Provider2",
  "date": "01.01.2017",
  "currency": "EUR",
  "tarif": [
    {
      "name": "Tarif1",
      "interval": "monthly",
      "parameter": [
        {
          "name": "Storage",
          "unit": "GB",
          "upper_bound": 100,
          "price": [
```

```
15              {
16                "minimum": 10,
17                "maximum": 100,
18                "pricing_unit": 10,
19                "price_per_unit": 10
20              }
21            ]
22          }
23        ]
24      }
25    ]
26  }
```

Listing 3: Unabhängiger nicht-binärer linearer Parameter

3. Unabhängiger, nicht-binärer, nicht-linearer Parameter:
In *Listing 4* wird ein unabhängiger, nicht-binärer, nicht-linearer Parameter dargestellt, diesmal als Beispiel die Anzahl der CPU-Instanzen. Es fällt auf, dass es sich in diesem Fall um einen Parameter handelt, der ohne jegliche Einheit auskommt, weshalb dieses Attribut hier ignoriert werden kann. Um darzustellen, dass es sich um einen nicht-linearen Parameter handelt, werden dem Parameterobjekt mehrere Preisobjekte hinzugefügt, eines für jedes Intervall, in dem der entsprechende Preis gilt. In diesem Beispiel ist eine Höchstmenge von 16 CPU-Instanzen möglich, signalisiert durch den Wert des Attributs *upper_bound*. Außerdem soll im Bereich von einer bis acht gewählten Instanzen ein anderer, teurerer Preis verlangt werden als bei neun bis 16 Instanzen. Dazu werden zwei Preise angegeben, einen für jedes Intervall. Der erste Preis deckt dabei das untere Intervall ab, gekennzeichnet durch den Minimalwert von eins und den Maximalwert von acht. Das restliche Intervall wird vom zweiten Preis mit dem Minimum von neun und dem Maximum von 16 abgedeckt. Dabei ist zu beachten, dass die Preise eines Parameters den vollständigen Bereich von Minimum bis Maximum abdecken müssen. Das Minimum eines Preises muss immer dem Maximum des vorherigen Preises plus eine Einheitengröße entsprechen. In dem Fall aus Listing 4 dürfte beispielsweise das Minimum des zweiten Preises nicht erst bei 10 anfangen, da sonst eine unspezifizierte Lücke entstehen würde.

```
1  {
2    "name": "Provider3",
3    "date": "01.01.2017",
4    "currency": "EUR",
5    "tarif": [
6      {
7        "name": "Tarif1",
8        "interval": "monthly",
9        "parameter": [
10         {
11           "name": "CPU-Instances",
12           "upper_bound": 16,
13           "price": [
14             {
15               "minimum": 1,
16               "maximum": 8,
17               "pricing_unit": 1,
18               "price_per_unit": 10
19             },
20             {
```

```
21                "minimum": 9,
22                "maximum": 16,
23                "pricing_unit": 1,
24                "price_per_unit": 8
25              },
26            ]
27          }
28        ]
29      }
30    ]
31  }
```

Listing 4: Unabhängiger nicht-binärer nicht-linearer Parameter

4. Abhängiger, binärer Parameter:
Listing 5 zeigt zwei voneinander abhängige Parameter, wobei es sich beim ersten, einer integrierten Backup-, sowie einer Monitoringfunktion, um einen binären handelt. Um dessen Abhängigkeit vom zweiten Parameter des Tarifs zu modellieren, wird bei beiden das Attribut *bundle* mit dem gleichen Zahlenwert, in diesem Fall eins, versehen. Damit geht ebenfalls einher, dass jeder Parameter des Bundles denselben Preis pro Einheit aufweisen muss. Da eine Systemkonfiguration, welche einen dieser beiden Parameter enthält, automatisch auch den anderen enthalten würde, wird der Preis in diesem Fall nur einmalig berechnet. In diesem Beispiel würde der Tarif den Nutzer daher 20 Euro kosten und nicht 40.

```
1  {
2    "name": "Provider4",
3    "date": "01.01.2017",
4    "currency": "EUR",
5    "tarif": [
6      {
7        "name": "Tarif1",
8        "interval": "monthly",
9        "parameter": [
10         {
11           "name": "Backups",
12           "unit": "binary",
13           "upper_bound": 1,
14           "bundle": 1,
15           "price": [
16             {
17               "pricing_unit": 1,
18               "price_per_unit": 20
19             }
20           ]
21         },
22         {
23           "name": "Monitorin",
24           "unit": "binary",
25           "upper_bound": 1,
26           "bundle": 1,
27           "price": [
28             {
29               "pricing_unit": 1,
30               "price_per_unit": 20
```

```
31              }
32            ]
33          }
34        ]
35      }
36    ]
37  }
```

Listing 5: Abhängiger nicht-binärer nicht-linearer Parameter

5. Abhängiger, nicht-binärer, linearer Parameter:
Wie ein abhängiger, nicht-binärer, linearer Parameter in einem Anbieterprofil modelliert wird, wird in *Listing 6* gezeigt. In diesem Fall sind die Anzahl der verfügbaren CPU-Instanzen und die Menge des RAMs voneinander abhängig. Wie im Beispiel zuvor ist dies durch das denselben Zahlenwert aufweisende Attribut *bundle* abgebildet. Ebenso weisen wieder alle zum Bundle gehörenden Parameter denselben Preis pro Einheit auf. Ein wichtiger Punkt, welcher bei mehreren abhängigen, linearen Parametern berücksichtig werden muss, ist, dass die Intervalle der Preise übereinstimmen müssen. In diesem Beispiel ist jede CPU-Instanz jeweils an 200MB RAM geknüpft, beide Werte wachsen also proportional im gleichen Maße. Der Parameter CPU-Instanzen kann im Bereich von eins bis 16 gewählt werden, bei einer Einheitengröße von 1, die Größe des RAMs im Bereich von 200 MB bis 3200MB, bei einer Einheitengröße von 200MB. Im Verhältnis zur Einheitengröße entspricht das in beiden Fällen der gleichen Intervallgröße von 16. Die Minimalmenge des Preises jedes Parameters muss also, wenn sie durch die entsprechende Einheitengröße geteilt wird, immer dieselbe sein. Das Gleiche gilt für die jeweilige Maximalmenge. Würde in diesem Beispiel der Maximalwert für den RAM lediglich bei 3000MB liegen, die mögliche Maximalgrenze der CPU-Instanzen jedoch gleich bleiben, würde das Verhältnis zwischen den Parametern nicht mehr stimmen. Für eine Menge von 16 CPU-Instanzen wäre dann keine passende Menge an RAM mehr möglich. Es ist zudem möglich, anstatt nur mehrere lineare Parameter zu verknüpfen, sowohl lineare als auch binäre Parameter innerhalb eines Bundles zu definieren. Ein nicht binärer Parameter muss dazu lediglich den Preis der kleinstmöglichen Konfiguration aufweisen. Dies entspricht dem Produkt des Minimalwertes des ersten Preisintervalls und dessen *price_per_unit*.

```
1  {
2    "name": "Provider5",
3    "date": "01.01.2017",
4    "currency": "EUR",
5    "tarif": [
6      {
7        "name": "Tarif1",
8        "interval": "monthly",
9        "parameter": [
10         {
11           "name": "CPU-Instances",
12           "upper_bound": 16,
13           "bundle": 1,
14           "price": [
15             {
16               "minimum": 1,
17               "maximum": 16,
18               "pricing_unit": 1,
19               "price_per_unit": 10
```

```
20              }
21            ]
22          },
23          {
24            "name": "RAM",
25            "unit": "MB",
26            "upper_bound": 3200,
27            "bundle": 1,
28            "price": [
29              {
30                "minimum": 200,
31                "maximum": 3200,
32                "pricing_unit": 200,
33                "price_per_unit": 10
34              }
35            ]
36          }
37        ]
38      }
39    ]
40  }
```

Listing 6: Abhängiger nicht-binärer linearer Parameter

6. Abhängiger, nicht-binärer, nicht-linearer Parameter:
Abhängige, nicht-binäre, nicht-lineare Parameter stellen den letzten und komplexesten Fall dar und werden in *Listing 7* beschrieben. Wieder werden die Anzahl der CPU-Instanzen an die Menge des RAMs durch denselben Wert des Attributs *bundle* geknüpft. Ebenfalls gleich bleibt das Verhältnis der Einheitengrößen, von 200MB RAM pro CPU-Instanz. Neu ist jedoch, dass jeder Parameter nun zwei Preise aufweist. Dabei muss jeder Parameter die gleiche Anzahl an Preisen besitzen und für jedes Preisintervall des einen Parameters muss jeder andere Parameter ein entsprechendes Äquivalent der gleichen Größe und dem gleichen Preis pro Einheit besitzen. Dementsprechend hat in diesem Fall der Parameter RAM ein Preisintervall von 200MB bis 1600MB, welches das Gegenstück zum Intervall von 1 bis 8 der CPU-Instanzen darstellt. Selbiges gilt für das RAM-Intervall von 1800MB bis 3200MB und das CPU-Intervall von 9 bis 16. Auch hier ist wieder wichtig, wie schon beim unabhängigen, nicht-binären, nicht-linearen Parameter, dass zwischen den Preisintervallen keine Lücken existieren.
Auch hier ist es wiederum möglich, nicht-lineare und binäre Parameter zu verknüpfen. Dabei ist es nicht nötig, dem binären Parameter mehrere Preisintervalle zuzuweisen, es genügt der Preis des ersten Preisintervalls.
Da abhängige nicht-binäre Parameter immer über die gleichen Preisintervalle, inklusive gleicher Preise verfügen müssen, ist es dementsprechend nicht möglich, und auch nicht gewollt, eine Abhängigkeitsbeziehung zwischen linearen und nicht-linearen Parametern zu modellieren.
Zur Veranschaulichung wurde das Bundle im Beispiel um den binären Parameter *Monitoring* ergänzt.

```
1  {
2    "name": "Provider6",
3    "date": "01.01.2017",
4    "currency": "EUR",
5    "tarif": [
```

```
6       {
7         "name": "Tarif1",
8         "interval": "monthly",
9         "parameter": [
10          {
11            "name": "CPU-Instances",
12            "upper_bound": 16,
13            "bundle": 1,
14            "price": [
15              {
16                "minimum": 1,
17                "maximum": 8,
18                "pricing_unit": 1,
19                "price_per_unit": 10
20              },
21              {
22                "minimum": 9,
23                "maximum": 16,
24                "pricing_unit": 1,
25                "price_per_unit": 8
26              }
27            ]
28          },
29          {
30            "name": "RAM",
31            "unit": "MB",
32            "upper_bound": 3200,
33            "bundle": 1,
34            "price": [
35              {
36                "minimum": 200,
37                "maximum": 1600,
38                "pricing_unit": 200,
39                "price_per_unit": 10
40              },
41              {
42                "minimum": 1800,
43                "maximum": 3200,
44                "pricing_unit": 200,
45                "price_per_unit": 8
46              }
47            ]
48          },
49          {
50            "name": "Monitoring",
51            "unit": "binary",
52            "upper_bound": 1,
53            "bundle": 1,
54            "price": [
55              {
56                "pricing_unit": 1,
57                "price_per_unit": 10
58              }
59            ]
60          }
```

```
61          ]
62        }
63    ]
64 }
```

Listing 7: Abhängiger nicht-binärer nicht-linearer Parameter

4.3 Schwächen des Modells

Mit Hilfe des vorgestellten Modells lässt sich nun jede der auf Grundlage der Datenerhebung bestimmten Parameterklassen vollständig abbilden. Da diese Klassifizierungen auf Basis der real auf *PaaSfinder.org* vorkommenden Anbieter entstanden sind, ist es im Prinzip möglich, für jeden einzelnen ein passendes Anbieterprofil zu erstellen. Allerdings zeigen einige Fälle auch Nachteile des aktuellen Modells auf. So weisen einige Parameterabbildungen Redundanzen auf, um die Berechnung in der Implementierung zu vereinfachen. Beispielsweise weist ein binärer Parameter die Tatsache, dass er binär ist, innerhalb des Attributs *unit*, durch den Wert *binary* aus. Zusätzlich dazu wird allerdings auch das Attribut *upper_bound* mit dem Wert eins belegt, was ein binärer Parameter im Grunde impliziert.
Des Weiteren ist bei jedem linearen Parameter der Inhalt des Attributs *upper_bound* immer identisch mit dem *maximum* des zugehörigen Preises.
Neben möglicher Redundanz besteht zudem das Problem, dass in einigen Fällen die JSON-Dokumente mit den Preisprofilen durch die vielen 1:n Beziehungen sehr schnell sehr groß werden können. Gerade nicht-lineare Parameter können, falls ein Anbieter auf viele, kleine Preisintervalle setzt, zu langen Dateien führen. Entscheidet sich ein Anbieter, die Intervalle, in denen ein Preis gilt, sehr klein zu halten, im Extremfall nur eine Einheitengröße pro Intervall, ist die Modellierung sehr aufwendig. Sind zudem mehrere nicht-lineare Parameter voneinander anhängig, steigt die Dateigröße zusätzlich, da jeder Parameter über die gleiche Anzahl von Preisobjekten verfügt.
Zudem lassen sich gegenwärtig Enumerationstypen nicht direkt darstellen. Sind beispielsweise für den Parameter *Supportlevel* die Ausprägungen *Community* und *Premium* verfügbar, so müssen dafür zwei verschiedene Parameter erstellt werden.
Eine weitere Schwäche besteht bei abhängigen nicht-binäre Parametern, welche nicht proportional zueinander wachsen. So kann beispielsweise eine einzelne CPU-Instanz in Kombination mit 20GB Hintergrundspeicher angeboten werden, zwei Instanzen jedoch mit 50GB. In diesem Fall müssten für beide Kombinationen jeweils ein eigener Tarif erstellt werden. Dennoch ist es letzten Endes maßgebend, dass all diese Anbietertarife modelliert und in das System eingebracht werden können.

5 Implementierung

5.1 Grundkonzept

Der zweite Teil dieser Arbeit beschäftigt sich mit der praktischen Erstellung einer Webseite, mit deren Hilfe Anwender die Preise einzelner Cloud-Anbieter vergleichen können. Der Nutzer soll hierbei die Möglichkeit haben anzugeben, welche Spezifikationen das von ihm

gewünschte Cloud-System besitzen soll, woraufhin ihm eine Liste aller infrage kommenden Angebote präsentiert werden soll. Mit Hilfe des vorher entworfenen Modells sollen die günstigsten zu erwartenden Preise der einzelnen Anbieter für die gewünschte Konfiguration errechnet werden. Damit der Nutzer auf einen Blick sehen kann, welcher Anbieter das für ihn beste Preis-/Leistungsverhältnis bietet, soll die Liste der infrage kommenden Anbieter zusätzlich im Preis aufsteigend sortiert werden.
Die Konfigurationsmöglichkeiten, die dem Nutzer dabei für seine Abfrage auf der Weboberfläche zur Auswahl stehen, sollen sich aus den aktuell vorhandenen Preisprofilen ergeben und dynamisch generiert werden. Dies hat zum einen den Vorteil, dass auf der Seite ausschließlich Parameter zur Auswahl stehen, die auch tatsächlich für die aktuellen Anbieterprofile relevant sind. Andernfalls wären Situationen möglich, in denen Werte zwar ausgewählt werden können, allerdings keinerlei Auswirkungen auf das Ergebnis haben, da sie schlicht für kein Preisprofil von Relevanz sind. Ein weiterer, weitaus größerer Vorteil ist allerdings die Tatsache, dass die Anwendung dadurch weitaus flexibler gehalten werden kann. Wie bereits erwähnt ist die Auswahl der geeigneten Parameter für den Preisvergleich schwierig, da stets ein Mittelweg zwischen Vollständigkeit und Benutzbarkeit gefunden werden muss. Zusätzlich können jederzeit neue Anbieter hinzukommen, sowie bestehende Anbieter ihre Preisstrukturen verändern. Durch die dynamische Gestaltung der Anwendung ist man davon unabhängig, da Änderungen und Ergänzungen an den Profilen automatisch in der Anwendung berücksichtigt werden.

5.2 PaaSfinder.org

Die entstandene Webanwendung soll Teil der bereits bestehenden *PaaSfinder*-Anwendung werden und folglich auch auf den gleichen Technologien aufbauen. Das aktuelle System nutzt das Webframework *Sinatra*, welches die Erstellung von Webseiten mit Hilfe der Skriptsprache *Ruby* ermöglicht. Zur persistenten Speicherung der Anbieter- und Preisprofile wird das NoSQL-Datenbankmanagementsystem *MongoDB* verwendet. Da die Profile im JSON-Format vorliegen und *MongoDB* Dokumentstrukturen verwendet, die dem JSON-Format sehr ähnlich sind, lassen sich die Daten ohne Probleme in die Datenbank einlesen, und performant wieder auslesen. Im Vergleich zu SQL-Datenbanken ist dazu kein Mapping zwischen den Relationen der Datenbank und den Objektstrukturen der Anwendung notwendig.

Der Aufbau der bisherigen Anwendung erfolgt nach dem klassischen *Model-View-Controller-Prinzip*. Im Model werden die Objektpräsentationen der Anbieter- und Preisprofile definiert, deren Struktur analog zu der Struktur der JSON-Dateien aufgebaut ist. Die Controller verwalten das Routing der Anwendung und bestimmen, wie auf die einzelnen HTTP-Requests reagiert werden soll, auf welche Views der Nutzer weitergeleitet werden soll und welche Daten an die Views weitergegeben werden sollen.

5.3 Umsetzung

5.3.1 Das Model

Der erste Schritt der Implementierung war die Integration der entworfenen Datenstrukturen für die Preisprofile in das Model der Anwendung. Das Model wird um eine Pricing

Kategorie erweitert. Diese enthält vier neue Klassen, die zur vollständigen Abbildung der Profile benötigt werden. Diese vier Klassen stellen die Objektrepräsentationen der Datenobjekte *VendorPricing*, *Tarif*, *Parameter* und *Price* dar, die im vorherigen Kapitel vorgestellt wurden. Die Relationen der Klassen sind analog zur JSON-Struktur der Preisprofile aufgebaut und bilden dementsprechend ebenfalls eine Baumstruktur.

Zusätzlich zu diesen vier Datenklassen wird eine weitere Klasse *FormParameter* erstellt, welche zur dynamischen Erstellung der Weboberfläche verwendet wird. Objekte dieser Klasse repräsentieren leichtgewichtigere Versionen der Parameter-Objekte, reduziert auf die wesentlichen Informationen, welche das Webinterface benötigt.

Die erstellten Klassen des Models weisen dieselbe Struktur wie die JSON-Dokumente des Datenmodells auf. Zusätzlich dazu wird die Klassenstruktur dazu verwendet, den Aufbau der *Collections* der Datenbank festzulegen.
Zu diesem Zweck wird das Werkzeug MongoID verwendet, ein *Object-Document-Mapper* mit dessen Hilfe Dokumentstrukturen für MongoDB anhand bestehender Objektstrukturen erstellt werden können. Es genügt dabei vollkommen, wenn JSON-Dokumente im entsprechenden Format vorliegen, um daraus automatisch die benötigten Datenbank-Collections zu erstellen, welche beim Auslesen direkt wieder die passenden Objekte bereitstellen.
Im vorliegenden Fall werden demnach zwei Collections erstellt. Zum einen die Tabelle *FormParameter* und zum anderen die Tabelle *VendorPricing*, basierend auf dem Wurzelelement des Datenmodells, welches ebenfalls die untergeordneten Daten der *Tarif*, *Parameter* und *Price* Klassen enthält.
Um sicherzustellen, dass die eingelesenen JSON-Dateien dem Format der Model-Klassen korrekt entsprechen, bietet MongoID die Möglichkeit, Validierungen für die einzelnen Felder der Klassen hinzuzufügen. Dadurch kann zum einen festgelegt werden, ob ein Attribut zwingend erforderlich ist oder ob es theoretisch weggelassen werden kann. Beispielsweise ist das Attribut *name* der Klasse *Parameter* zwingend erforderlich, um diese unterscheiden zu können, und darf dementsprechend nicht weggelassen werden. Im Gegensatz dazu ist das Feld *unit* derselben Klasse nicht zwingend erforderlich, da ein Parameter nicht zwingend eine Einheit benötigt, sondern rein über die Anzahl definiert wird.

Die JSON-Dokumente, welche die Preisprofile der Anbieter enthalten, befinden sich in dem separaten Ordner *pricings*, so wie sich die bisher von *PaaSfinder.org* verwendeten Anbieterprofile in einem eigenen Ordner *profiles* befinden. So lassen sich leicht bestimmte Anbieter finden um Änderungen vorzunehmen, veraltete Daten zu entfernen und natürlich auch komplett neue Profile hinzuzufügen.
Um die bisherigen Anbieterprofile in die Datenbank einzulesen, wird ein *Task* verwendet, der jedes Mal manuell ausgeführt wird, wenn Änderungen an den Daten vorgenommen werden. Damit die neu hinzugekommenen Preisprofile ebenfalls nach Änderungen in die Datenbank aufgenommen werden können, wurde der dafür zuständige Task *db:seed* um die zusätzliche Funktion *pricing* erweitert, welche die Profile in die Datenbank-Collection *VendorPricing* einliest. Sie funktioniert analog zur bisherigen Funktion, welche die einzelnen Vendorprofile überträgt. Zuerst wird der aktuelle Inhalt der Datenbanktabelle gelöscht. Anschließend wird jedes im Ordner *pricings* enthaltene JSON-Dokument geparst und, sofern die Struktur des Dokuments korrekt ist, jeweils ein neuer Datenbankeintrag erstellt. Es genügt hierbei vollkommen, den Wurzelknoten der Datenstruktur anzugeben, in diesem Fall *VendorPricing*, da MongoID automatisch die JSON-Struktur auf die erstellte

Klassenstruktur des Models überträgt.

Wie bereits erwähnt soll die Preisvergleichs-Webseite dynamisch anhand der bei den Anbietern vorkommenden Parameter erstellt werden. Um zu verhindern, dass bei jedem Seitenaufruf alle Preisprofile aufs Neue durchsucht werden, um die Menge aller Preisparameter zu erhalten, wird diese Liste beim Ausführen des db:seed Tasks zusätzlich erstellt. Nach dem Füllen der VendorPricing-Collection wird diese daher einmal vollständig durchlaufen und jeder gefundene Parameter wird in die separate Tabelle *FormParameter* eingetragen. Mehrfach vorkommende Werte werden dabei nur einmal hinzugefügt. Da einige Attribute der Parameter-Klasse für die Erzeugung des Eingabeformulars nicht von Bedeutung sind, wie zum Beispiel *bundle*, *upper_bound*, sowie die referenzierten Preis-Klassen, werden diese in der FormParameter-Collection ignoriert. Durch die zusätzliche Tabelle entsteht zwar eine gewisse Redundanz und zusätzlicher Speicherbedarf durch mehrfach gespeicherte Daten, der Overhead, der bei jedem Seitenaufruf durch das Erzeugen des Formulars entsteht, wird jedoch, vor allem bei einer großen Anzahl an Anbietern, drastisch reduziert.

5.3.2 Die Controller

Dem Controller *main.rb*, welcher für die Verarbeitung und Weiterleitung von HTTP-Requests zuständig ist, wurde ein weiteres Routing hinzugefügt. Der GET-Request an die URL *www.paasfinder.org/pricing* führt auf die neue Seite für den Preisvergleich. Für den Fall, dass der Nutzer sich bereits auf der Seite befand und eine PaaS-Konfiguration angegeben hat, wird diese dem Request als Parameter mitgegeben und an den zusätzlichen Controller *price_calculator.rb* weitergereicht. Dieser Controller ist dafür zuständig, anhand der eingegebenen Konfiguration auszuwählen, welche Tarife welcher Anbieter infrage kommen, zu berechnen, welche Kosten dabei auf den Nutzer zukommen würden, und zu bestimmen, wie das angebotene Gesamtpaket des jeweiligen Tarifs letztendlich aussehen würde.

Zu Beginn der Berechnung wird bestimmt, wie viele Parameter vom Nutzer ausgewählt worden sind, um später überprüfen zu können, ob der gefundene Tarif auch tatsächlich den Ansprüchen genügt. Da nur Tarife angeboten werden sollen, welche die Anforderungen des Anwenders auch vollständig erfüllen, werden Angebote, die weniger Parameter bieten als gewünscht, ignoriert. Danach wird ein Hashobjekt erstellt, welchem später die Resultate hinzugefügt werden. Anschließend wird auf der Suche nach den passenden Angeboten jeder Tarif jedes vorhandenen Anbieterprofils durchgegangen. Für jeden Anbieter wird dabei eine Hashmap *tarifs* angelegt, in welcher die Tarife gespeichert werden, die für das Ergebnis relevant sind. Die Informationen über jeden Tarif werden im Objekt *tarifInfo* gespeichert. Dazu zählen der Name des entsprechenden Tarifs, dessen Preis, sowie die exakte Systemkonfiguration, die der Nutzer zu erwarten hat. Zusätzlich wird jedem Tarif eine Variable *pricing* zugeordnet, in welcher später Schritt für Schritt die entsprechenden Einzel-Preise kumuliert werden.
Im nächsten Schritt werden die einzelnen Parameter eines Tarifs überprüft, wobei hier eine Fallunterscheidung durchgeführt wird. Es wird kontrolliert, ob es sich um einen unabhängigen Parameter handelt, das Attribut *bundle* also nicht gesetzt ist, oder ob er nur in Verbindung mit anderen Parametern verfügbar ist. In diesem Fall hat das Attribut *bundle* der entsprechenden Parameter jeweils denselben Integer-Wert.

Im einfachen Fall, wenn ein unabhängiger Wert vorliegt, wird überprüft, ob der Parameter

des Tarifs einer derer ist, die vom Nutzer ausgewählt worden sind. Ist dies der Fall, wird zusätzlich kontrolliert, ob die vom Anwender gewünschten Dimensionen im Rahmen des Tarifs liegen oder ob sie das Attrbiut für die Maximalmenge, *upper_bound*, überschreiten. Beispielsweise soll in dem Fall, dass der Nutzer 64GB RAM verlangt, der Tarif des Anbieters aber nur maximal 32GB bieten kann, der Tarif nicht weiter berücksichtigt werden. Liegt die gewünschte Menge jedoch unterhalb der Maximalmenge, kann mit der Preisberechnung fortgefahren werden. Liegt die gewünschte Menge im Gegensatz dazu unterhalb des minimal möglichen Wertes des Tarifs, wird dieser nicht aussortiert, sondern die verlangte Menge wird auf den nächst möglichen Wert aufgerundet. Andernfalls würden Tarife, welche die Erwartungen des Anwenders zwar erfüllen, aber in mindestens einem Punkt sogar überschreiten, aus dem Raster fallen. Diese Designentscheidung bietet den Vorteil, dass Cloud-Konfigurationen, die der Nutzer auf der Weboberfläche wählen kann, nicht exakt als Tarif in dieser Form existieren müssen, sondern lediglich als Minimalanforderung angesehen werden.

Da jeder Parameter über mehrere Preise verfügen kann, die beispielsweise unterschiedliche Mengenintervalle abbilden, wird nun jeder Preis einzeln überprüft.
Im einfachsten Fall handelt es sich bei dem gewählten Parameter um einen binären, also einen Wert, der entweder gesetzt ist oder nicht. Dies wird mit Hilfe des Parameter-Attributs *unit* erkannt. Trägt dieses den Wert *binary*, wird die *pricing*-Variable schlicht um den Preis dieses Parameters (*price_per_unit*) erhöht.
Handelt es sich jedoch nicht um einen binären Wert, wird eine kompliziertere Berechnungsmethode angewendet. Ein Parameter kann, wie bereits erwähnt, über mehrere Preisobjekte verfügen. Beispielsweise kann so ein Mengenrabatt modelliert werden, indem für ein bestimmtes Intervall ein bestimmter Preis pro Einheit vorgesehen ist, ab einem festgelegten Schwellenwert dann jedoch ein günstigerer Preis in Kraft tritt, der wiederum für das nächste Intervall gilt. Dabei wird jedes Intervall von einem Preisobjekt repräsentiert, welches den Gültigkeitsbereich mit Hilfe der Attribute *minimum* und *maximum* abgrenzt. Zunächst wird die eingegebene Menge des Parameters, mit Hilfe der *roundValue*-Methode, auf die nächstgrößere, durch dessen Einheitengröße (*pricing_unit*) teilbare Zahl gerundet. So wird, wenn beispielsweise zwölf Einheiten gewollt sind, der Parameter aber nur in Zehnerschritten angeboten wird, der Wert auf 20 gerundet.

Im nächsten Schritt wird für jedes Preisobjekt überprüft, ob die gerundete Menge größergleich dem Minimum und kleiner-gleich dem Maximum des Objektes ist und damit innerhalb des Intervalls liegt. Sollte dies nicht der Fall sein, kann der entsprechende Preis ignoriert werden. Triff dies jedoch zu, ist der aktuelle Preis für diese Menge gültig und kann berechnet werden. Der letztendliche Preis entspricht dabei:

$$price\_per\_unit * gerundeter_Wert / pricing\_unit$$

Enthält ein Parameter also beispielsweise zwei Preisintervalle und der gewählte Wert liegt im höheren der beiden, wird der Preis dieses Intervalls verwendet. Der dabei errechnete Preis wird anschließend zum *pricing*-Wert im Hashobjekt des aktuellen Anbieters hinzuaddiert. Da es sich um eine unabhängige Kenngröße handelt, erzeugt jede davon eigene Kosten, welche den Gesamtpreis erhöhen. Anschließend wird, da auf der Weboberfläche ebenfalls angezeigt werden soll, welche Parameter die angegebene Konfiguration im Detail zu bieten hat, die aktuelle Kenngröße einer Hashtabelle *paramHash* hinzugefügt, welche wiederum Bestandteil von *tarifs* ist. Dabei dient der Name des Parameters als Schlüssel,

der dazugehörige Wert entspricht dem gerundeten Eingabewert.

Sollte die Fallunterscheidung ergeben, dass ein Parameter eines Tarifs ein gesetztes *bundle*-Attribut besitzt und damit nicht unabhängig von anderen ist, wird anders vorgegangen als bei unabhängigen Werten. Einzelne Bundles werden über ihre Nummer identifiziert. Für gefundene Bundles wird das Hashobjekt *bundles* erzeugt, welches als Schlüssel die Bundlenummer verwendet. Der dazugehörige Wert beinhaltet ein weiteres Hashobjekt, mit den Schlüsseln *parameters*, *price* und *interval*. Der Wert *parameters* verweist auf ein Array von Parameter-Objekten, in welchem alle Parameter, die im besagten Bundle vorkommen, gespeichert werden. Der Schlüssel *price* referenziert den Gesamtpreis, den das Bundle bei der aktuellen Konfiguration verursachen würde und wird mit dem Wert 0 initialisiert. Der Wert, welcher über den Schlüssel *interval* erreichbar ist, wird später verwendet, um die Anzahl der einzelnen Parameter zu berechnen.

Wird ein Parameter als Teil eines Bundles gefunden, wird dem *bundles*-Objekt ein neuer Eintrag mit der entsprechenden Bundlenummer hinzugefügt, sofern dieser noch nicht vorhanden ist. Der Parameter selbst wird dem Array unter dem Schlüssel *parameters* hinzugefügt. Die Berechnung der Preise anhand ihrer Intervalle erfolgt im Grunde analog zur Preisberechnung unabhängiger Parameter. Ein Unterschied besteht allerdings darin, dass bei unabhängigen Werten die einzelnen Preise berechnet und anschließend aufaddiert wurden, da jeder Parameter separat Kosten verursachte. Bei Bundles ist dies nicht der Fall. Wird beispielsweise ein Parameter ausgewählt, der Teil eines Bundles ist, sind daran automatisch andere Parameter geknüpft, die der Nutzer zwar nicht ausgewählt hat, welche aber trotzdem Bestandteil des Angebots sind. Wählt der Anwender nun eine zusätzliche Kenngröße aus, die aber im Grunde schon Bestandteil des Tarifs ist, soll dies logischerweise keine Auswirkungen auf den Preis haben, da sich nur die nachgefragte Konfiguration ändert, nicht aber die angebotene. Aus diesem Grund wird immer wenn ein Parameter, der Teil eines Bundles ist, gefunden wird, der Preis separat für diesen berechnet. Der finale Preis ergibt sich dann aus dem Maximum aller vorkommenden Preise. Jedes Mal wenn also die Kosten für einen einzelnen Parameter berechnet werden, wird überprüft ob der aktuell im Hashobjekt *bundles* für das aktuelle Bundle hinterlegte Preis, welcher zu Beginn 0 beträgt, kleiner ist als der soeben berechnete. Ist dies der Fall, so wird der bisherige Preis durch den Neuen ersetzt. Dies hat zur Folge, dass nur der Parameter, welche die höchsten Kosten verursacht, den Preis bestimmt. Da jeder Parameter eines Bundles über den gleichen Preis pro Einheit verfügt und die Intervalle jedes Parameters gleich groß sind, ergibt sich somit automatisch der Preis für die kleinstmögliche Konfiguration, die alle Ansprüche des Nutzers erfüllt.

Auf die gleiche Art und Weise wird auch die letztendliche Gesamtkonfiguration des Bundles bestimmt. Jedes Mal, wenn der Preis eines Parameters berechnet wird, wird nicht nur überprüft, ob bereits ein höherer Preis errechnet wurde, sondern auch, in welchem Preisintervall sich der Parameter mit dem höchsten Wert befindet. Da nicht-binäre Parameter eines Bundles immer in gleichem Maße wachsen, lässt sich anhand dieses Wertes bestimmen, wie viele Einheiten jeder Kenngröße die aktuelle Konfiguration enthält. Bei jeder Preisberechnung wird zusätzlich überprüft, wie vielen Einheiten des Parameters die gewählte Menge entspricht. Eine Menge von 40 und einer Einheitengröße (*pricing_unit*) von 10 beispielsweise entspricht vier Einheiten. Ist dieser Wert größer als der aktuell in *bundles* unter *interval* festgeschriebene Wert, welcher wie auch der Preis mit 0 initialisiert wird, dann wird der alte durch den neuen Wert ersetzt. Nach dem Durchlaufen aller Parameter kann mit Hilfe dieses Wertes berechnet werden, welche Ausmaße die Gesamtkonfiguration

hat, indem die Einheitengröße jedes enthaltenen Parameters mit dem Intervall multipliziert wird.
Ist die Berechnung aller Kenngrößen abgeschlossen, werden alle Parameter eines Bundles dem Hashobjekt *paramHash* hinzugefügt. Dabei handelt es sich um dasselbe Objekt, in welchem bereits die unabhängigen Kenngrößen gespeichert wurden. Als Schlüssel wird hier jeweils der Name des Parameters verwendet, der Wert entspricht der Anzahl der Einheiten.
Zu guter Letzt wird überprüft, ob der aktuelle Tarif für den Nutzer infrage kommt. Dies ist der Fall, wenn die Anzahl der vom Anwender gewünschten Parameter mindestens so groß ist wie die Anzahl der gefundenen passenden Parameter. Der Fall, dass die Anzahl der gefundenen Werte größer ist als die der gesuchten, tritt auf, wenn gebündelte Parameter involviert sind und somit automatisch auch ungewollte Kenngrößen hinzukommen können. Wird der Tarif als passend bewertet, wird das Objekt *tarifs* den Resultaten hinzugefügt. Diese Überprüfung wird erst am Ende vorgenommen, um ein zweimaliges Durchlaufen der Preisprofile zu verhindern. Nachdem die Tarife aller Anbieter vollständig durchsucht worden sind, werden die Ergebnisse dem Preis nach aufsteigend sortiert und der Controller gibt schlussendlich die Resultate an den aufrufenden Main-Controller zurück, welcher diese wiederum an die View *pricing.erb* weiterreicht. Dabei ist zu beachten, dass, sollte aktuell der Fall auftreten, dass mehrere Tarife eines einzelnen Anbieters passen, der Übersicht halber nur der günstigste davon übergeben wird.

5.3.3 Die View

Die Weboberfläche der Anwendung ist in zwei Bereiche untergliedert. In der oberen Hälfte befindet sich das Eingabeformular, in dem die Spezifikation des gewünschten Cloud-Systems vorgenommen werden kann. Die daraufhin gefundenen Resultate werden in der unteren Hälfte angezeigt. Das Webframework *Sinatra* setzt zur Darstellung von Internetseiten nicht auf die Verwendung von reinen HTML-Dateien, sondern auf sogenannte *Embedded-Ruby-Dateien(ERB)*, welche sowohl klassisches HTML, als auch Ruby-Code enthalten können. So lässt sich beispielsweise die Darstellung bestimmter Elemente steuern. Ruby läuft dabei serverseitig und kann dazu verwendet werden, die Struktur der Seite dynamisch an den Inhalt anzupassen. Das Webinterface für diese Anwendung besteht aus einer einzigen View, der *pricing.erb*, welche über die URL */pricing* erreicht wird.

Für die Darstellung des oberen Teils der Seite, welche alle vorhanden Parameter inklusive Eingabefelder enthalten soll, werden zunächst alle Objekte der Datenbanktabelle *FormParameter* benötigt. Die Tabelle enthält, wie bereits erwähnt, leichtgewichtigere Versionen der Preisparameter, welche einzig und allein für die Erstellung des Eingabeformulars benötigt werden. Sie erlaubt einen performanten Seitenaufbau, ohne dass bei jedem Seitenaufruf die kompletten Preisprofile aller Anbieter durchsucht werden müssen. Innerhalb eines HTML-Formulars, das einen GET-Request auf die URL */pricing*, inklusive angehängter Anfrageparameter, auslöst werden die aktuell vorhandenen Kenngrößen angezeigt. Dazu wird die *FormParameter*-Tabelle vollständig durchlaufen und dabei für jedes Objekt unterschieden, ob es sich um einen binären oder einen nicht-binären Wert handelt, da dies Einfluss auf die Darstellung, sowie die Verarbeitung hat. Dies geschieht mit Hilfe des Attributs *unit* des *FormParameter*-Objekts. Je nachdem um was für eine Art von Wert es sich handelt, hat das Auswirkungen darauf, welcher Typ von Inputfeld neben dem Namen des Parameters erscheint. Bei einem binären Wert hat der Nutzer nur die Möglichkeit

festzulegen, ob dieser für ihn von Relevanz ist oder nicht. Daher wird in diesem Fall eine Checkbox angezeigt, bei welcher der Anwender einen Haken setzen kann, sollte ihm der entsprechende Parameter wichtig sein. Handelt es sich dagegen um einen nicht-binären Wert, wird ein Textfeld angezeigt, in welches der Nutzer die gewünschte Anzahl eingeben kann. Das Inputfeld verwendet dabei den HTML5-Type *number*, wodurch die Eingabe auf Zahlenwerte beschränkt wird. Durch einen zusätzlich festgelegten Minimalwert von 0 wird entsprechend verhindert, dass negative Werte angegeben werden können. Bei nicht-binären Parametern, wird außerdem neben dem Eingabefeld die entsprechende Einheit des Parameters angegeben, sofern diese vorhanden ist.
Die in der View definierte Weboberfläche ist vollständig unabhängig vom eigentlichen Inhalt der Preisprofile. Änderungen an Datensätzen in der Datenbank, wie beispielsweise neu hinzugefügte Parameter, haben direkte Auswirkungen auf das Webinterface und werden automatisch übernommen. Das hat den Vorteil, dass bei sich ändernden Daten, keinerlei Änderungen am Code vorgenommen müssen.

Der zweite Teil der Weboberfläche ist für die Darstellung der Ergebnisse zuständig, die nach dem Abschicken des Formulars vom Controller geliefert werden. Sobald der Submit-Button betätigt wird, wird ein GET-Request an die URL */pricing* gesendet, dem alle vom Nutzer ausgewählten Daten als Anfrageparameter angehängt worden sind. Der Controller *main.rb* gibt zuerst die Parameter an eine Instanz der *PriceCalculator*-Klasse weiter und nimmt daraufhin das Ergebnis entgegen, welches wiederum der View als Klassenvariable *results* in Form eines Hashobjektes mitgegeben wird.
In der pricing.erb Datei wird mit Hilfe dieser Resultate der Inhalt des unteren Teils der Oberfläche aufgebaut. Bei jedem Eintrag von *results* handelt es sich um einen für den Nutzer infrage kommenden Tarif, inklusive Tarif- und Anbietername, der Systemkonfiguration, den dieser Tarif bietet, sowie natürlich dem Preis. Das Hashobjekt wird nun einmal vollständig durchlaufen, wobei jedes Element in einen separaten div-Container gepackt wird, welcher die Ausgabe der gesamten Informationen formatiert. Die einzelnen Resultate erscheinen dadurch übereinander aufgelistet, immer nur ein Tarif pro Zeile. Da der Inhalt des Hashobjekts bereits im Controller nach Preis sortiert wird, ist ein Sortieren hier nicht mehr notwendig. Die Tarife werden automatisch im Preis aufsteigend angezeigt.

Das Design des Webinterfaces orientiert sich am Design der bisherigen *PaaSfinder.org*. Im oberen Bereich befindet sich der Titel der aktuellen Seite, in diesem Fall *PaaS Price Comparison*, inklusive einer Unter-Überschrift, welche die Seite beschreibt. Es wird ebenfalls auf das CSS- und Javascript-Framework *bootstrap* zurückgegriffen, welches unter anderem mit vielen vorgefertigten CSS-Klassen das Styling der Webseite unterstützt. Der Titel beispielsweise wird mit Hilfe der Klasse *Jumbotron* vergrößert und zentriert. Für die Anordnung der div-Container auf der Webseite wird das Grid-System von Bootstrap verwendet, welches das horizontale und vertikale Positionieren von Elementen vereinfacht. Dabei lässt sich mittels der Klasse *row* ein div-Container als horizontale Reihe definieren, welche in zwölf Bereiche, sogenannte *Columns* unterteilt ist. Jedem Element innerhalb dieses Containers kann nun, ebenfalls über eine CSS-Klasse, vorgeschrieben werden über wie viele dieser zwölf Bereiche es sich erstrecken soll. So besitzt beispielsweise jeder Container, der ein Resultat des Preisvergleich darstellt, die Klasse *row*. Dem Bereich für den Preis wird dabei nur eine *column* an Breite zugewiesen, dem Bereich der die Informationen über gebotene Konfiguration des Tarifs enthält, aber ganze fünf, da diese theoretisch auch länger werden können.

6 Fazit und Ausblick

Das Ziel dieser Arbeit bestand darin, eine bessere Vergleichbarkeit von PaaS-Angeboten zu ermöglichen und damit einen weiteren Schritt in Richtung einer Standardisierung von PaaS-Systemen zu gehen. Zu diesem Zweck sollte zum einen ein einheitliches Datenmodell entworfen werden, das in der Lage ist, die Preisstrukturen der PaaS-Anbieter abzubilden. Ein wesentlicher Punkt, der dabei berücksichtigt wurde, war, dass das Modell ausreichend flexibel gestaltet werden sollte. Aufbauend auf diesem Modell sollte im Anschluss eine Webanwendung für die Seite *PaaSfinder.org* implementiert werden, die es ermöglicht, Preise von PaaS-Angeboten zu vergleichen.
Um zu erörtern, wie unterschiedlich die Preismodelle im PaaS-Bereich sind und was bei der Modellierung alles beachtet werden muss, wurden die Preisdaten von 71 Anbietern analysiert. Dabei wurde festgestellt, dass die Preismodelle sich sehr stark unterscheiden. Im Schnitt kommt mit nahezu jedem Anbieter ein weiterer Parameter ins Spiel, der Einfluss auf den Preis eines Angebots haben kann. Da die Berücksichtigung jedes einzelnen dieser Parameter sich negativ auf die Usability des Preisvergleichs auswirken würde, musste eine Auswahl getroffen werden. Mit neu hinzukommenden Anbietern und sich verändernden Tarifen kann es jedoch möglich sein, dass diese Auswahl in Zukunft geändert werden muss. Daher wurden, anstatt die einzelnen Parameter zu berücksichtigen, deren Eigenschaften untersucht. Auf Grundlage dieser Untersuchung entstanden sechs verschiedene Klassifikationen von Parametern, in welche sich alle aktuell von Anbietern verwendete Parameter einteilen lassen. Basierend auf dieser Klassifikation wurde anschließend ein Modell entworfen, das alle Fälle vollständig abbilden kann, unabhängig von der Semantik des Parameters und dem zugrunde liegenden Preismodell. Damit lässt sich für jeden Anbieter ein JSON-basiertes Preisprofil erstellen.
Im Zuge der Implementierung wurde das entstandene Datenmodell in eine Webanwendung für *PaaSfinder.org* integriert. Diese erzeugt anhand der vorliegenden Preisprofile der Anbieter dynamisch eine Weboberfläche, auf der der Nutzer seine gewünschte PaaS-Konfiguration auswählen kann. Darauf basierend bekommt er anschließend passende Ergebnisse preislich sortiert präsentiert.
Für die Zukunft sind noch einige Verbesserungen an Modell und Anwendung möglich. So findet aktuell die verwendete Währung des Anwenders keine Beachtung. Es existieren Ruby Bibliotheken (sogenannte *gems*), welche die Währungsumrechnung anhand von Life-Daten umsetzt. Die *gem money* kann für die Implementierung dieses Features verwendet werden. Zudem können bestehende Redundanzen im Datenmodell weiter reduziert werden. Außerdem ist zu erwähnen, dass bisher bei weitem nicht alle Anbieterprofile in die Anwendung integriert wurden. Zum aktuellen Zeitpunkt befinden sich zu Testzwecken sechs ausgewählte Profile im *pricings*-Ordner der Anwendung.
Die Anwendung wurde so konzipiert, das Änderungen und Ergänzungen an den Anbieterprofilen vorgenommen werden können, ohne Ändnderungen an der Software durchführen zu müssen. Es genügt, die Profile zu erstellen und die Datenbank zu aktualisieren. Die Anwendung verwendet dann automatisch die aktuellen Daten.
Das löst zwar nicht das Problem, dass es schwierig ist, die passenden Parameter für die Anwendung auszuwählen, da sich deren Wichtigkeit schnell ändern kann. Die Auswahl erfolgte in dieser Arbeit als *Best Guess* und dient letztlich dem Tauglichkeitsnachweis des Vorgehens. Für eine optimale Auswahl der Parameter müsste eine größere Menge realer Benutzer befragt werden, was den Rahmen und das Ziel dieser Arbeit gesprengt hätte. Das gewählte Vorgehen ermöglicht allerdings dem Administrator der Anwendung, zukünftige Änderungen mit möglichst wenig Aufwand zu übernehmen.

Literatur

[1] Danamma M.Bulla and Dr. V R. Udupi. Cloud Billing Model: A Review. *International Journal of Computer Science and Information Technologies*, 5(2), 2014.

[2] Stefan Kolb and Guido Wirtz. Towards Application Portability in Platform as a Service. *Proceedings of the 8th IEEE International Symposium on Service-Oriented System Engineering (SOSE)*, 2014.

[3] Sururah A. Bello and Gazali Abdul Wakil. Flexible Pricing Models for Cloud Services. *Transactions on Networks and Communications*, 2(5), 2014.

[4] May Al-Roomi, Shaikha Al-Ebrahim, Sabika Buqrais, and Imtiaz Ahmad. Cloud Computing Pricing Models: A Survey. *International Journal of Grid and Distributed Computing*, 6(5), 2013.

[5] Hongyi Wang, Qingfeng Jing, Bingsheng He, Zhengping Qian, and Lidong Zhou. Distributed Systems Meet Economics: Pricing in the Cloud. In *HotCloud '10*, 2010.

Ich erkläre hiermit gemäß §17 Abs. 2 APO, dass ich die vorstehende Bacherlorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Bamberg, den 01.12.2017 Thomas Aberle

Parameter	Abs. Häufigkeit	Rel. Häufigkeit	Parameter	Abs. Häufigkeit	Rel. Häufigkeit	Parameter	Abs. Häufigkeit	Rel. Häufigkeit
RAM	26	70,27%	Data-Transfer	2	5,41%	Task-Priority	1	2,70%
CPU	22	59,46%	Private VLAN	2	5,41%	LDAP/SAML/AD	1	2,70%
Storage	22	59,46%	IPv6	2	5,41%	HTTPS	1	2,70%
Bandwidth	12	32,43%	Number of projects	2	5,41%	App collaboration	1	2,70%
Backups	11	29,73%	Number of teams	2	5,41%	Cron jobs	1	2,70%
Supportlevel	10	27,03%	Operation Systems	2	5,41%	GPU	1	2,70%
Scalability	9	24,32%	Role based access control	2	5,41%	Dedicated inbound IP	1	2,70%
SSL	9	24,32%	API-Calls	2	5,41%	Number of services	1	2,70%
VPN	6	16,22%	Number of users	2	5,41%	Number of processes	1	2,70%
Logging	5	13,51%	Public cloud	2	5,41%	SSH access	1	2,70%
Public IP	5	13,51%	On premise	2	5,41%	Portal users	1	2,70%
Monitoring	4	10,81%	Self-healing	1	2,70%	Private deployment platform	1	2,70%
Managed DNS	4	10,81%	Isolation	1	2,70%	Workflow automation	1	2,70%
Requests	4	10,81%	Clustering	1	2,70%	Git integration	1	2,70%
MongoDB	4	10,81%	Secure remote access	1	2,70%	Slack integration	1	2,70%
Application idling	4	10,81%	Enterprise ready	1	2,70%	Audit logs	1	2,70%
Metrics	3	8,11%	Self-healing	1	2,70%	Fine grained ACLs	1	2,70%
Eigene Domain	3	8,11%	PCU-Mindestabnahme	1	2,70%	Multi region deployments	1	2,70%
MySQL	3	8,11%	Custom host names	1	2,70%	Self hosted console	1	2,70%
PostgreSQL	3	8,11%	SLA-Reaktiv	1	2,70%	Virtual machines	1	2,70%
Loadbalancing	3	8,11%	SLA-Proaktiv	1	2,70%	Uptime guarantee	1	2,70%
Private Cloud	3	8,11%	SAN	1	2,70%	Failover	1	2,70%
Number of apps	3	8,11%	Global IP	1	2,70%	Continuous integration	1	2,70%
						raining	1	2,70%

Tabelle 1: Parameterverteilung