The Open Tariff Model – Towards Efficient Cost Comparison of Public Cloud Services

Jörg Gottschlich, Oliver Hinz
Chair of Information Systems III, Electronic Markets
Department of Economics and Law
TU Darmstadt

June 1, 2016

With the continuing growth of the public cloud service market, enterprises increasingly face more complex and more frequent decisions to optimize their IT infrastructure. When it comes to the question of on-premise versus cloud deployment and provider selection in the latter case, cost figures are needed for fact-based decisions. In the public cloud case, enterprises needs to comprehend and apply a diversity of tarification systems to compare potential cloud providers. This effort slows down decision processes and reduces the number of considered alternatives. Therefore, we present an Open Tariff Model which provides a unified cost model to support cost estimates for IT deployments in the public cloud by mapping diverse provider tariffs into a common structure. Such a model helps to quickly compare provider costs for cloud deployments or provides input to analyses of the cloud service market, among other applications like cloud brokerage.

1. Introduction

Along with the growing market for cloud services [Moore, 2015], new challenges for the sourcing of IT resources arise. Different service models like Infrastructure-, Platform- and Software-as-a-Service (IaaS, PaaS, SaaS) enable a more fine-grained decision about in- or outsourcing of IT components than before. Instead of either operate a whole IT system on-premise or by a contractor, service-oriented architectures enable enterprises to decide *for each layer* of a solution stack to either build it within their own systems or buy it from a vendor and integrate it into their architecture. While this creates more degrees of freedom for best-of-breed approaches (i.e. to choose the best solution for every single aspect of a business problem) it also increases the complexity of IT procurement and governance. For example, when a company creates an IT solution

for its customer support processes, it has a range of deployment options: 1) Host a licensed software on hardware in-house, 2) lease virtual machines (VMs) via IaaS to install the application, possibly combined with PaaS services like managed databases or 3) subscribe to a turn-key SaaS solution which provides convenient operations but usually also limited customization possibilities. Each option has its pros and cons, and evaluating all these – and potentially more – options causes high effort.

In addition, standardization increases the agility of IT solutions while reducing vendor lock-ins and path dependencies. Enterprises can more easily extend or move their infrastructure independent of its current deployment space which might create new opportunities to optimize IT provisioning. With these developments on a more fragmented and dynamic market, enterprises face a higher number of solution alternatives and shortened decision cycles to optimize their infrastructure layout.

This generates a need for decision support to efficiently compare vendors and their products and to tap the full potential of cloud computing as technology. Hence, some approaches for cloud service selection have been proposed (e.g. [Rehman et al., 2011,Garg et al., 2013, Repschlaeger et al., 2013]). However, still lacking is a comprehensive cost model to describe full costs of a cloud solution in the public cloud. Previous works focus on service features and quality, but disregard cost or use simple hourly prices. Cloud providers created diverse and complex tariff models involving many parameters to estimate the cost of a complete setup including computing nodes, storage, traffic, network components and others. For a projected system setup, such a model would avoid having to understand and evaluate different tariff systems when comparing offers from different providers.

As an example, take the tariffs of two providers: Amazon Web Services (AWS) [Amazon Web Services, 2015] and ProfitBricks [ProfitBricks, 2015]. AWS offers compute instances as a fixed bundle of CPU cores, memory and, partly, storage at a certain price. For example, the *t2.medium* instance, at a price of $0.052 per hour in their US East data center, has 2 CPU cores, 4 gigabyte (GB) memory and no storage included while the *m3.xlarge* contains 4 cores, 15 GB memory and 2x40 GB local SSD drives at a price of $0.266 per hour. With these bundled offers, a consumer who needs more memory also receives (and pays for) more CPU cores. Moreover, a direct price comparison between providers is difficult if they have slightly different bundles of CPU, RAM and storage. Other providers, like ProfitBricks, charge per component. Consumers can choose the desired number of cores, size of memory and storage space. The price is composed of the hourly unit prices (US data center): $0.018 per core per hour + $0.0053 per GB memory per hour and $0.04 per GB per month for block storage. Centron, another provider offering free combination of components, offers 250 GB of free traffic [Centron GmbH, 2015] while the other two offer not more than 1 GB. On the other hand, Centron charges in-coming and out-going traffic, while the other two only count out-going traffic. These are just three providers and some of their specific regulations. When we examined a larger sample of public cloud tariffs (cf. Section 4), we found that pricing among provider tariffs differs along a multitude of dimensions for realistic infrastructure setups:

- Hourly prices for virtual machine instances depending on CPU cores, size of RAM, and internal storage
- Extended storage options (e.g. block storage, SSD, redundant storage, content delivery networks, backup storage)
- Traffic (e.g. different volumes, different regions, different networks)
- Datacenter Location (e.g. US, Europe, Asia, South America)
- Operating system and software applications (e.g. Windows and other proprietary software causing license fees, Ubuntu Linux, RedHat Enterprise Linux)
- Network components (e.g. firewalls, load balancers, static IP addresses)
- Utilization (e.g. 24/7 operations, peak demands, nightly jobs)
- Deployment period (hours to years, e.g. test system, project infrastructure, core business infrastructure)
- Discounts (e.g. volume discounts, prepay discounts)

Unit prices can differ between providers on each of these dimensions without necessarily revealing a generally inferior or superior providers in terms of total cost over all dimensions. Depending on the resource needs of an IT setup, a holistic cost comparison of provider tariffs needs to consider all these factors as any of them could be a cost driver for a certain application purpose that turns an interesting offer based on e.g. hourly VM pricing into an inferior offer (e.g. if traffic pricing is too high for the intended (high traffic) application) – or vice versa. But evaluating the total costs of all parts causes a high effort for a request-for-proposals process or cost comparison of in-house solutions versus cloud deployment.

From a provider's perspective, it is also increasingly difficult – particularly for small vendors – to stand out in the competition and differentiate their offers in a market moving quickly from innovation to commoditization. With increasing competition and substitute solutions to their products, their communication and acquisition efforts rise. Especially when they are not able to communicate their advantages efficiently, customers might focus solely on price as decision criterion which could lead to a lemon market where a quality-focused differentiation strategy fails to be effective [Akerlof, 1970].

Up to date, we found no model in literature which is able to store the pricing information of the cloud service market such that comprehensive market analyses can be conducted. For research, such a model provides detailed price and cost information for analyses and decision frameworks. For practitioners, such a model enables efficient vendor comparison of cloud deployments for fact-based business decisions.

In order to compute prices for different provider tariffs, we decided to create a tariff model that formulates a superset of all tariff properties we find in the market and hence enable a unified specification of provider tariffs. By mapping the providers' tariffs into our generic model, we are able to compute prices for any resource combination, which

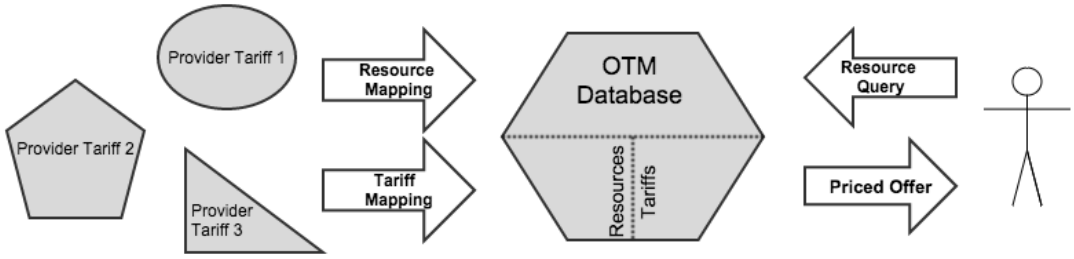

Figure 1: Tariff Unification by the OTM

can be expressed using the language of the model (Figure 1). Therefore, we introduce the Open Tariff Model (OTM) which is extensible for yet even unknown types of cloud resources, but still provides appropriate details for price computation such as pricing conditions, multi-part pricing and discounts (cf. 4). Thus, our model strives to unify provider offers for easy vendor comparison and to deliver precise cost estimates for IT infrastructure decisions very efficiently (e.g. for a vendor selection or make-or-buy decision).

We considered other ways of achieving price information for decision support. Most preferable would be a unified API to query any provider for prices of a resource bundle (e.g. similar to the approach in [Amato & Venticinque, 2013]). However, due to missing established standards of resource specification for such requests for proposals (RFPs) and appropriate technical API implementations, it is no feasible option. Some providers offer calculators on their websites to request prices for resource descriptions which could be queried for price estimates, but they are usually difficult to operate automatically and one would still need to translate a resource request into the provider's own format. Also, for the many providers who do not offer price calculators, this solution is not feasible. Hence, we decided to create a generic tariff model and take the one-time effort to map provider tariffs into and use it as a common basis for price computations (Figure 1).

To achieve this goal, the paper continues as follows: After introducing preceding and related efforts to this problem in Section 2, we share the findings from our tariff study in Section 4. We discuss our methodology in Section 3. Then, we introduce the model components and their relations (Section 5) and show how to apply them to model a common provider tariff in Section 6 for evaluation purposes. Finally, we conclude our work in Section 7 and comment on potential future improvements.

2. Related Work

In literature, we find a considerable amount of cloud service description and comparison approaches with different scopes and targets. Most of the works focus on technical or feature comparison of cloud services starting with simple performance benchmarks up to extensive broker approaches. However, many papers disregard cost at all or use simple hourly prices to model cost. While this might be sufficient for a simple comparison of

virtual machines, it neglects important cost drivers for real world infrastructure setups, such as storage or network features.

One of the few works that puts a strong focus on advanced price and cost modeling is El Kihal, Schlereth, and Skiera (2012). They suggest two methods to make IaaS price comparison more transparent: Hedonic pricing which decomposes the price into contributing values of the tariff's characteristics and a new-developed pricing plan comparison which aims to identify the most favorable requirement profile for each provider compared to its competitors [El Kihal et al., 2012]. They validate their approaches in an empirical study and use their results to compare providers. However they only consider VM pricing and dismiss other important tariff factors like network resources. In addition, they start from a predefined set of VM configurations and prices from a fixed number of providers. This information thus still needs to be compiled from tariff information. The OTM would provide such detailed information and hence complement their work on the input side and offer possibilities to enhance such tariff analyses.

Other works focus on a top-down approach and suggest a market-oriented model, i.e. a model that allows for negotiation between consumers and providers (e.g. [Siebenhaar et al., 2011, Roovers et al., 2012, Wang et al., 2013]). For example, Roovers et al. (2012) suggest an auction model with resource descriptions based on tagging to express asks on which providers can bid [Roovers et al., 2012]. However, these approaches have in common that they require an interactive role of the provider to enable interactive pricing. Moreover such a business model faces a startup problem in a two-sided market, i.e. the so-called chicken and egg problem, which ususally hinders the adoption in the market in the start.

Another type of related work develops complex decision frameworks for cloud service adoption, quite frequently by applying a multi-criteria decision method such as an Analytical Hierarchical Process (AHP) (e.g. [Menzel et al., 2013]). Garg et al. (2013), for example, provide a comprehensive catalogue of a benchmarking and measurement environment for cloud providers [Garg et al., 2013]. Their work is based on the service measurement index (SMI) by The Cloud Service Measurement Initiative Consortium (CSMIC) [The Cloud Service Measurement Initiative Consortium (CSMIC), 2011], which is used to develop a Service Measurement Index Cloud framework SMICloud [Garg et al., 2011]. SMICloud contains sixteen different aspects modeled as key performance indicators, which are extracted from the SMI and which are highly relevant for computing in cloud environments. The paper however disregards important aspects like fixed prices, hidden pricing components and different tariffs for a single cloud provider.

Patiniotakis et al. (2014) extend the SMICloud model (by Garg et al.) on different levels by introducing more qualitative factors such as *Contracting Experience* or *Technical competency of the support employees* and suggest a recommender approach for cloud services which distinguishes between precise quantitative service attributes (e.g. response time) and imprecise qualitative, more intuitive characteristics (e.g. reputation). With their approach they want to enable users to explicitly express fuzzy levels of requirements and apply a fuzzy multi-criteria decision making method to arrive at a flexible preference-based ranking of cloud services [Patiniotakis et al., 2014].

Repschlaeger et al. (2013) use a survey among IT executives to define priority weights

for an AHP for non-core business applications. Thus, they quantify the importance of factors such as flexibility, costs and performance for IaaS provider selection in "general" business contexts. However, their approach stays abstract and while they provide detailed evaluation criteria, they do not involve actual market data for their cloud service comparison.

While these approaches target the problem of cloud service comparison, they neglect the aspect of data provisioning. For a regular application in decision making, the approach of ad-hoc research is not feasible. Instead, a sustainable database of market data is necessary for efficient decision-making. The OTM enables the construction of such a database by providing a flexible generic model to store the providers' products and pricing information in a comparable way.

The Distributed Management Task Force (DMTF) has published an extensive specification of IaaS resources with their Cloud Infrastructure Management Interface (CIMI) specification [DMTF, 2013]. This standard is targeted at technical operability between different cloud providers and provides very detailed properties such as operating status attributes of resources. Thus, for the purpose of product comparison, their model is very extensive and thus carries large overhead. In addition, standardization processes cause inertia. For a cost model serving business decisions, which strive to optimize infrastructure on the quickly evolving cloud service market this could oppose a major disadvantage.

The Open Cloud Computing Interface (OCCI) standard for Infrastructure of the Open Grid Forum has a similar focus on technical interoperability [Metsch & Edmonds, 2011] and while more light-weight than the CIMI specification, it misses certain attributes important for commercial resource comparison. In addition, the latest release of the standard is dated April 2011 and its future access of adoption is uncertain. Therefore, we strive for a light-weight bottom-up standard model instead of a comprehensive top-down model (cf. Section 3.2) to support an efficient vendor comparison.

3. Methodology

3.1. Design Science

This paper follows the Design Science paradigm which constitutes an approach rooted in the engineering sciences [Simon, 1996]. The goal is to solve a certain problem by providing an artifact design and evaluating its suitability to solve the problem. Hevner et al. (2004) provide several guidelines to systematically achieve this goal which we follow in the development of our proposed model concept [Hevner et al., 2004].

The introduction and the theoretical foundation in Section 1 and 2 show the relevance of the topic in the context of research. The importance of solving a business problem such as IT infrastructure sourcing efficiently, as pointed out in Section 1, becomes evident in the higher complexity of sourcing options and the shorter decision cycles imposed by increasing market evolution and product flexibility that come with cloud computing.

Our artifact, the Open Tariff Model, addresses this problem and provides a solution (Section 5) which is the result of the rigor search based on market research and induction

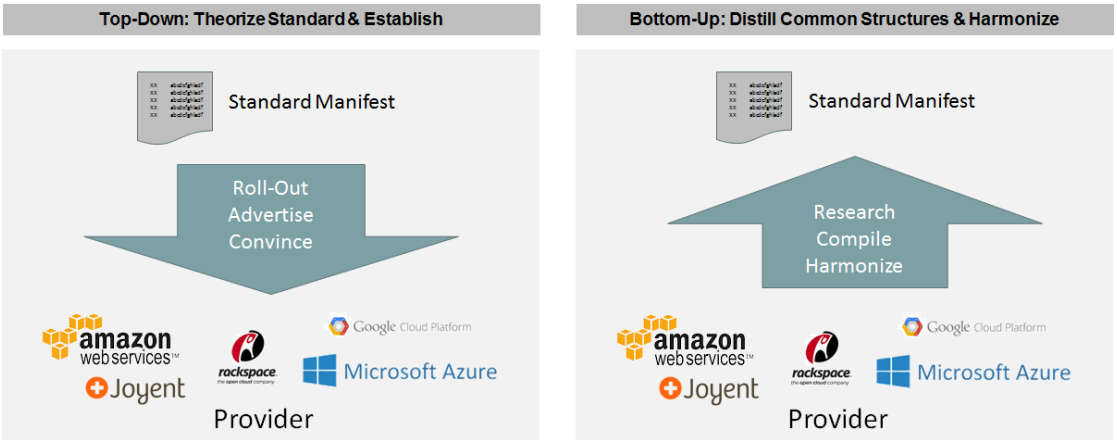

Figure 2: Two ideal models of standardization efforts

from a tariff analysis (see 3.2) for a solution to the identified problem of cloud service cost estimation. We evaluate our design and show its application in Section 6. In our concluding remark (Section 7), we summarize our research contribution. All in all, this paper communicates our research to foster discussion as well as further research and applications in this area.

3.2. Standardization Approach

To create the OTM, we conducted a review of several different provider tariffs and identified common structures to destill the design of the OTM (see Figure 1). Providers' tariffs can differ quite a lot in detail, but usually there is a good share of similarity, which lowers the effort of implementing a generic model.

This work seeks to develop a data model which is able to capture the cloud service market data on an appropriate level of detail, down to product features relevant for application scenarios. Therefore, we have to standardize the actual products and market conditions to be able to capture them in a common model. Figure 2 shows two different archetypal approaches to standardization: Top-down standardization which involves a normative process of defining theoretical constructs and then impose them on the problem domain (the cloud service market in our case) and bottom-up which strives to define standard constructs by identifying common structures and abstracting them into a generic form. This latter method is comparable to the Grounded Theory approach [Glaser & Strauss, 2009] which iteratively refines structures identified in collected data.

The latter method works well to extract a standardization model from such a complex domain with the advantage that it does by design match the market requirements. A special advantage for a service comparison application, as intended here, is that the model generation already focuses on the comparable product features. Product specifications unique to one vendor only, which are thus not comparable, have lower priority

(but can still be added). In addition, a model derived with this inductive approach does not suffer from acceptance barriers – as some top-down standards do – as it embraces what already exists and ensures compatibility.

On the downside, this approach has a passive position towards market development, i.e. it does not push for harmonization of services in the market, but instead takes the complexity and heterogeneity as given facts to which it adjusts. Normative standards have the chance to align structures and improve service comparability by affecting the service definition by providers. Nevertheless, we are confident that for the purpose of market transparency, the inductive approach has advantages in terms of efficiency, flexibility and effectiveness as it can quickly react on changes in the market.

When conducting the tariff review, we started with a few heterogeneous tariff examples and built a first draft of the model from those. We then checked it against further tariff examples to see if the draft holds. Upon evolvement of new requirements, we adjusted the model and checked it back with the first set of tariff examples. When the model worked for all chosen samples, we took some more samples to see if their requirements hold and continued this iterative refinement. In addition, we frequently checked for opportunities to reduce complexity. In the end, we built our model upon several hundred tariff schemes of around 30 providers of different regions and sizes, covering more than half of the market volume and all major providers [Preimesberger, 2015].

4. Market Overview from Tariff Research

Based on our tariff analyses, we identified some entities which allow to build a generic tariff model (compare Figure 3). To manage complexity, we start with a simple structure and divide the tariffs into two components: 1) The different cloud service types which a user can purchase with their specific features (which we call *Resources* in the model) and 2) The pricing part which describes how costs are calculated for these resources given a certain quantity and usage period and other usage-specific parameters (cf. 5.2.1, Table 2).

As the market is constantly developing and changing, this is only a snapshot of the current situation. While the basic building blocks (e.g. VMs, Storage) are quite stable, the features of those services may extend and also providers work to introduce new kinds of services in their struggle for competitive advantage. This section gives some insights on the complexity of cloud service tarification which helps to understand the subsequent model formulation.

4.1. Common Cloud Service Classes

While cloud services often differ between providers in technical details, we identified some common services classes providers usually offer. We will introduce those classes one by one below. Larger providers usually have several data centers geographically spread around the world with a concentration in North America and Europe. So in addition to the problem of provider choice, subsequently, consumers also have to choose a data center location which has consequences for issues like data protection legislation

and connection latency. Also, some providers do not offer all of their products in every data center.

4.1.1. Compute Instances (Virtual Machines)

The core part of IaaS cloud services is a virtual server instance which usually consists of one or more CPU cores, a certain amount of memory (RAM) and, usually, some storage capacity. From a tarification perspective, there are providers offering bundled sizes of compute instances (e.g. Amazon Web Services, Google Compute Instances) with each having a fixed amount of cores, memory and storage. Other providers allow you to configure the desired amount of cores, memory and storage without pre-configured relations (e.g. ProfitBricks, CloudSigma) and charge per each of those components (hence we refer to them as *component-based* provider tariffs). Component-based tariffs have an advantage for users with skewed resource needs (e.g. needing a lot of RAM, but not so much processing power) as bundles usually scale in all dimensions and hence with more memory you would also need to purchase more CPU cores. Bundled tariff providers encountered this disadvantage by offering "high mem" or "high compute" instances which put on overweight on the respective component to better suit these skewed demands. For a pricing comparison of compute instances between component-based and bundled tariff providers, we need to first compose the bundled instance from the components (CPU, RAM and possibly storage) to arrive at a comparable price for the instance.

Pricing itself usually involves an hourly fee. Upfront payments for a certain time period to reduce the variable pricing per hour are usual, e.g. one or three year plans with a payment at the beginning and then a reduction in hourly prices for this period.

4.1.2. Operating System (OS)

An operating system is necessary to run a VM. Usually, a certain Linux flavor is included at no additional cost. But for premium OS versions, a surcharge applies, specifically when deploying Microsoft Windows, but also some Enterprise Linux versions from vendors like SuSE or RedHat. Windows versions often come with additional software packages installed such as Microsoft SQL Server or IIS (web server) which results in higher charges. If there are charges, they are usually a surcharge on the hourly rate of the VM, sometimes having different prices depending on the total number of cores in a VM (e.g. Google).

When configuring the VM at a provider, the user can select from a choice of OS images which subsequently will be deployed on the VM.

4.1.3. Storage

Storage services come in wide variety of purposes and technologies which causes differences in tarification. VMs can have *internal* storage which appears like a built-in hard disk from an OS perspective. It is created and destroyed with the VM itself. *Block* storage is attached over the network and mounted as a volume. It usually is independent

from the existence of a VM and serves for permanent data storage, but volatile variants are available as well. *Snapshot* storage saves state freezes of VMs at a certain point in time which can be restored or cloned for other purposes. *Image* storage enables a user to upload customized images of VMs, e.g. a pre-configured Linux variant for a certain business use, which the user can deploy on new VMs. For *backup* purposes, providers sometimes offer specific storage products with features specifically suited for this purpose (e.g. geographically redundant, cheap but with slow access).

The exact storage technology should theoretically be of no interest to the user, as (s)he procures a service and not a device. But if technology affects features of the services, it needs to be considered. For example, some tariffs offer fast solid state drives (SSDs) compared to slower magnetic drives – as the former are more expensive than the latter, but also offer a higher performance, we need to consider those features in a tariff comparison.

The most common pricing scheme for storage is per capacity per time (e.g. GB/-month). Prices differ with the outlined purpose specialization mentioned above and the technology involved. Apart from the storage capacity, the input/output operations per second (IOPS) determine the performance of storage systems as it denotes how much operations the device is able to process in a certain time [Nobel, 2011]. Some providers, such as AWS, use I/O operations as a price-determining factor and put a price on e.g. an amount of 1 million I/O operations [Amazon Web Services, 2015].

4.1.4. Data Transfer (Traffic)

Data Transfer (Traffic) is an essential element of every IT and cloud setup. Incoming traffic (*ingress*) is usually of no charge, but there are some exceptions (e.g. AWS charges incoming traffic when an external IP address is assigned to the interface [Amazon Web Services, 2015]). Outgoing traffic (*egress*) has to be distinguished into several scopes, depending on the target of the transmission. Within the same data center or data center zone traffic is usually free. Also if traffic stays within the provider infrastructure, e.g. accessing different services of the provider hosted at different locations, the out-going traffic often is handled as internal traffic even when spanning data center boundaries. However, most outgoing traffic pricing schemes either treat the whole Internet as one destination or differ by large geographic world regions (e.g. Asia/Pacific, Europe, North America). Pricing is either based on data center location or the destination of the connection – which means traffic cost either depend on cloud users' choice of data center location or on the geographic distribution of their clients.

A special case for traffic is a content delivery network (CDN) service. These are distribution networks for digital content, e.g. software or music, which are spread around the globe and specialize in providing a fast and reliable service of content delivery. If offered, this service usually has special traffic pricing. There are also independent (non-cloud) provider companies who offer such a service.

4.1.5. Network features

An IT infrastructure deployment needs network components to be complete. Cloud providers offer these services as accessories to the other infrastructure components. A *load balancer* helps to distribute incoming requests over a number of processing VMs to scale load and increase availability by redundancy. A *firewall* filters traffic by rules, limiting protocols, ports, sources or destinations, e.g. allow only web page requests to a web server. The pricing of both traffic processing components is based on deployment time, traffic processed, per rule, or a combination of those.

Another network component is a static IP address which assigns a fixed IP to a VM to make servers available under a stable address. They are usually charged on a time basis (e.g. per month), but often they are free when assigned to a running instance (e.g. [Amazon Web Services, 2015], [Google, 2015]).

Especially for auxiliary network services, there is a large diversity among providers who try to differentiate with advanced network services. For comparison purposes, we included the most common services here which offer comparable features. More specialized services should be added as they are adopted by market participants.

4.2. Pricing Conditions

Based on the tariff review, we identified some pricing conditions we find in the market and classify them to prepare the formulation of a common model. Like in other comparable markets, different pricing schemes increase the effort of price comparison. By integrating the components used by cloud providers in the model, we are able to compute cost using our common model and hence efficiently create cost comparisons for a certain cloud setup.

Due to their service nature, cloud products usually have a price based on time units (per hour, per month). A second dimension is the capacity of the service, e.g. for storage the size in gigabytes or a countable number of instances (e.g. one, two, three static IP addresses) – we summarize this second dimension with the term *quantity* of the cloud resource. There are also prices with both or none of these dimensions. If there is no quantity assigned with a price, we have e.g. only "EUR/hour" or if both quantity and time are missing, e.g. for one-time fixed fee, we would only have "EUR".

Next to on-demand pricing with a price per hour, we also find two-part tariffs, consisting of an upfront payment for a certain period and a (reduced) variable price per hour [Amazon Web Services, 2015]. Monthly or even annual payments with no hourly pricing also exist.

Apart from that, we also identified some conditions on pricing which we need to consider in order to select the correct price:

Applicability Some prices are only applicable if a certain amount of a resource is consumed. For example, having ranges from 0-5 gigabyte of traffic at price p_1 and 5-10 gigabyte at price p_2, when we determine the price for 8 gigabyte, the price would be p_2. So depending on the amount of traffic per month, we would need to apply a differ-

ent price. Consumption-based price differentiation is a frequent scheme also for other resource types.

Partitioned Pricing In contrast to applicability which denotes the price for the *total consumption quantity*, partitioned prices defined different prices for different *shares of* the total consumption. With the example above, we would split up the 8 gigabyte into $5 * p_1$ and $3 * p_2$ and hence apply a different price for each portion of the total quantity.

Booking Periods Not all tariffs are available on-demand and hence have a certain minimum booking period which affects the total cost as resources have to be paid for the total period. In exchange, commitment for a longer period usually lowers the variable price and hence, depending on the total deployment time of the cloud resources, brings a cost advantage compared to on-demand tariffs.

Minimum Charges Some tariffs have a minimum revenue. If the actual cost for resource consumption is above this amount, it has no effect, but for smaller setups it needs to be considered for tariff comparisons.

So cloud service pricing quite often involves processing of several prices per component, either in combined or alternative application.

4.3. Discounts

Discounts are relative price changes depending on conditions. We found conditions such as:

Spend A common discount condition is the achievement of a certain spend level (minimum revenue) within a certain time period (e.g. a month or a year)

Utilization Commonly, providers do charge no cost (or substantially less) to customers for compute instances which are not running, but are suspended. Because this decreases revenue, providers have an objective to increase uptime of customers. Discounts based on utilization, i.e. actual run-time of reserved resources, might stimulate the customer's willingness to extend run-times. Thus, they may provide a discount if the user reaches a certain range of monthly utilization. Google, for example, offers bands of 0-25%, 25-50%, 50%-75% and 75%-100% with respective discount percentages [Google, 2015].

Period commitment The commitment for a certain time period is quite common in business and enables vendors to spread their acquisition costs over a longer period and retain customers. Hence, some providers incentivize such an agreement with a discount.

Payment terms If customers are willing to pay in advance for a projected consumption, the provider has more certainty about his revenues and customer retention. Therefore, some providers grant discounts if the consumer pays in advance.

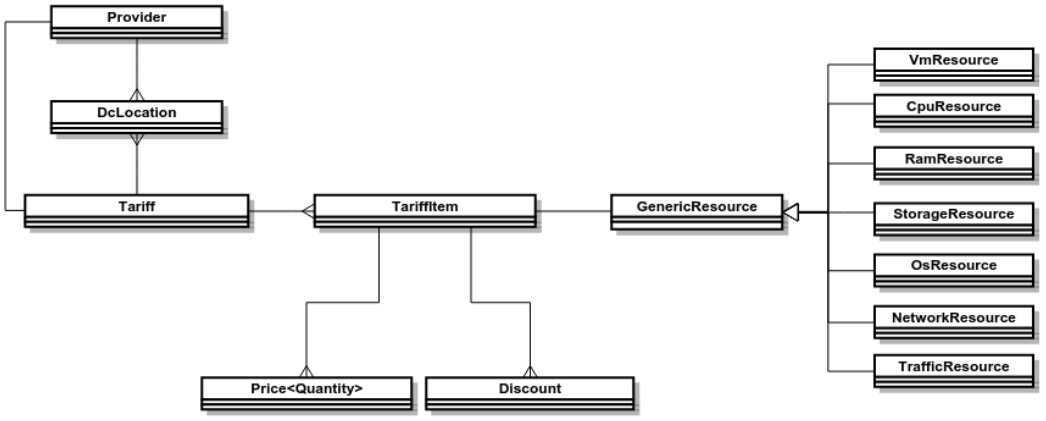

Figure 3: The OTM entity model

Providers define discounts on different levels. They can either apply to revenue with one specific cloud service (*resource* level), all services within one tariff (*tariff* level) or apply to the consumer's total revenue volume with the provider (*provider* level).

Thus, for evaluating potential discount amounts and net costs with a provider, the conditions have to be checked and the corresponding cost base has to be estimated. Given the uncertainty with the on-demand character of cloud services, assumptions have to be made. The model presented in this paper helps to quickly estimate optimistic or conservative scenarios to gain better insight in the effects of discounts on total cost at a certain provider.

5. Model Formulation

In this section, we introduce the actual model components and their relations, before we present the details of price and cost computation. Figure 3 provides an overview of the model. The central entity of our OTM is a `Tariff` entity which belongs to a `Provider` and a data center location (`DcLocation`). A `Tariff` holds a list of `TariffItem`s which connect one `Resource` with a list of applicable `Price`s and `Discount`s. We split the technical information about the *product* a provider offers and the commercial information about how the provider *prices* its product. While the product information is represented by the resource hierarchy, all pricing information is included in the `Price` and `Discount` entities. This allows for a clean separation of product and pricing and helps to update changes in prices independently from changes of the technical product itself.

As convention, we use over-lining for the $\overline{\text{upper bound}}$ of a range and underlining for the $\underline{\text{lower bound}}$. Boolean values are translated to numbers using 1 for *true* and 0 for *false*.

Table 1: `GenericResource` properties

Field	Description
name	Display name of the resource (e.g. "Storage")
subtype	Further classification for this resource type (e.g. block/snapshot storage)
validValues	The valid quantity values the provider offers for this resource, see App. A.2
quantity	Specifies an amount of the resource

5.1. Resource Model

In the OTM, we call all kinds of services a user can purchase from a cloud provider *resources*. To keep the model flexible and extendable, we model specific resources as more specialized descendants of a `GenericResource` type, which holds the properties common to all resources (cf. Table 1) and enables a common implementation of cost calculation for all resources. An extension of this hierarchy enables further specialization or introduction of new resource types without restrictions.

In addition to its `name`, each resource contains a `subtype` property to distinguish more specific kinds of a resource without introducing new classes for similar resources which have the same properties and actions. For example, specific storage types like snapshot storage (intended to store VM image states) or High-IOPS store (very fast storage) are both storage resources. In this case, the we would have two instances of the `StorageResource` entity, one with the `subtype` *snapshot* and the other with `subtype` *highio*. We can use a joint set of common storage properties to describe them (though with different values).

When describing a provider's offer, we need to store the available quantities of a certain resource (e.g. the user can choose from 1, 2, 4, 8 or 16 cores, but no value in between or the amount of storage must be between 10 and 1000 gigabytes). So to describe provider offers, `validValues` stores a discrete list or continuous range of valid values (cf. App. A.2) which holds the resource unit sizes offered by the provider. For example, a provider offers storage only in 10 gigabyte packages – so a consumer needs to purchase 30 gigabytes if he asks for 25 gigabytes.

If the resource entity stores a specific instance of a resource description, e.g. for an VM which has two cores and 16 gigabytes RAM, the `quantity` attribute holds the instantiated value, in this case two for CPU cores and 16 for RAM size in gigabytes.

Because the unit of measurement for those quantities differ by resource type, we implement a generic quantity model (cf. App. A.1) so we can handle the different units of measurement for resource quantities and prices in a unified way. In addition, we keep the model extensible for future resource types and their units of measurement. This quantity model also allows us to specify data units in megabyte, gigabyte, terabyte etc. and ensures correct conversion. Appendix A.1 explains the quantity model in detail.

During our market research, we identified some common resource classes (cf. 4.1)

which we model as specialized resources, i.e. as descendants of a `GenericResource`.

Computing `VmResource` models a bundled VM instance with a certain amount of CPU cores, RAM size and disk storage (if applicable). This is the resource used for providers offering bundled compute resources (such as Amazon Web Services or Google Compute Engine). For component-based providers, we define `CpuResource` and `RamResource`. These types model a CPU and RAM unit for providers which offer individual VM configuration. While `VmResource` is self-contained, we need to combine `CpuResource` and `RamResource` (and, potentially, a `StorageResource`) to model a usable compute instance.

OsResource contains information about available operating system options a provider offers within its tariff and keeps information such as OS family (Linux, Windows etc.), vendor and version.

StorageResource describes all different kind of storage space providers offer as services (e.g. internal, block, snapshot or backup storage).

TrafficResource contains information about the type and quality of traffic. Usually, depending on the direction and scope of traffic, different prices apply. Hence, if maximum precision is needed, we need to distinguish not only between incoming and outgoing traffic (*ingress* and *egress*), but also if the traffic is within or across boundaries such as the local data center, the world-wide infrastructure of the provider, the "open" Internet and which continents or regions. For this paper, we use a simplified specification which only differs between ingress and egress from and to the Internet. Because traffic within the provider's infrastructure is usually cheaper, this is a valid upper boundary for traffic costs. By extending the properties of `TrafficResource`, we can build complex data stream models to distinguish the exact flow of data traffic and apply individual prices on these different resource configurations.

NetworkResource models different network services such as firewall rules, static IP addresses or port forwarding services.

As discussed, these are the common resource types, we predefine from our market research. For new kinds of cloud services, we can easily inherit new resource types from `GenericResource` and model them with their specific properties to extend the OTM.

5.2. Commercial Model

Now that we introduced the technical part of the OTM, we turn to the modeling of costs. We introduce how we arrive at total tariff cost from single prices and how we apply discounts on cost.

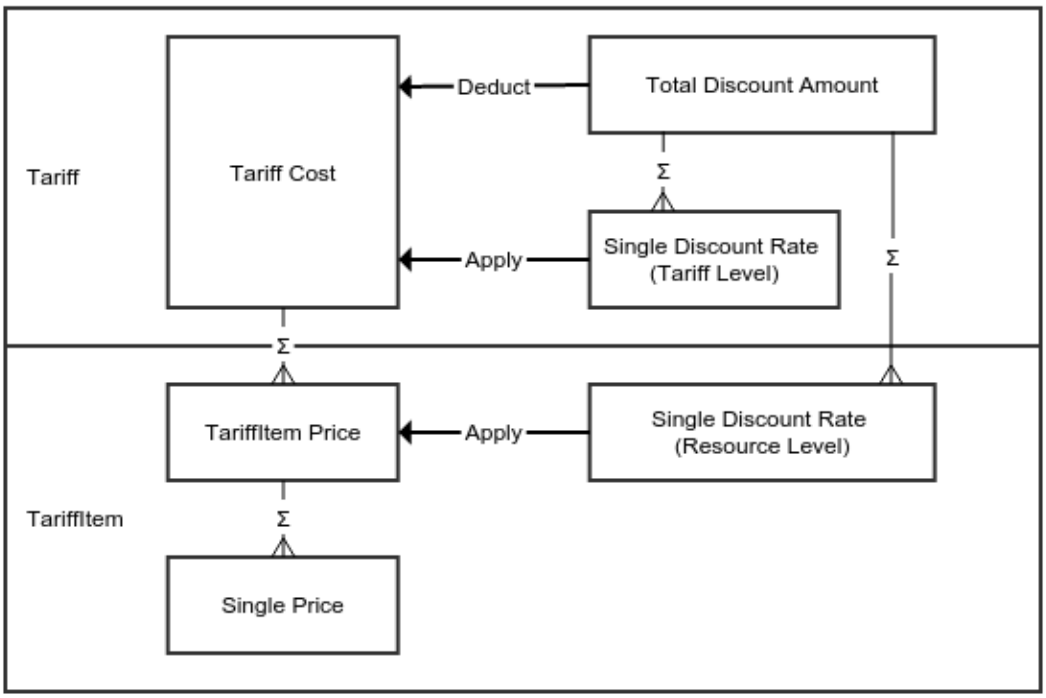

Figure 4: Cost Aggregation Scheme for OTM cost modeling

5.2.1. Cost Aggregation Scheme

A `Tariff` as the central entity of the OTM connects a product a certain provider offers in its datacenters with the prices including requirements such as minimum time or quantity and other conditions. Therefore, besides the attributes a `Tariff` holds, it keeps a list of `TariffItems` which connect one resource with the corresponding prices and, if applicable, discounts.

A `TariffItem` contain several prices or discounts for a resource to model the different pricing conditions (cf. 4.2). Therefore, a `Price` contains the specific conditions which apply to it. All prices assigned to a `TariffItem` will be summed up, with their value being computed according to their specific conditions (see 5.2.2). We model discounts in a similar way (see 5.2.3).

Figure 4 gives an overview of the cost aggregation scheme applied by the OTM. We start with the computation of all single prices p_i of a `TariffItem`, then aggregate the results to the total `TariffItem` cost c_j, on which we apply the discount factor and compute the absolute ***`Discount`*** *value on resource level* d_j (cf. 5.2.3). The sum of all c_j will result in the ***`Tariff`*** *cost before discounts* C_{T+} on which we apply the discount factor for ***`Discounts`*** *on tariff level* D. We add up the resulting discount values to the total discount value D and deduct it from C_{T+}, so we arrive at ***`Tariff`*** *cost after discounts*

C_T. Equation 1-3 reflect this computation for a `Tariff` of n `TariffItem`s.

$$C_T = C_{T+} - D \tag{1}$$

$$C_{T+} = \sum_{j=1}^{n} c_j \tag{2}$$

and

$$D = \sum_{j=1}^{n} d_j \tag{3}$$

For a cost estimate, we need to specify a deployment setup, i.e. define a *request*. Such a request consists of a list of k resources and some parameters which are listed in Table 2. The OTM enables us to determine the cost of a request for all tariffs stored with the model such that we can compare the total cost for each tariff for this request.

We need a target resource quantity and time for which the cost should be determined. We denote Q_k as the desired purchase quantity of resource k and T as the purchase period for all resources. Due to pricing conditions (e.g. a minimum booking period) for some resource, during calculation we might arrive at a different quantity or time than requested, so we introduce q_k or t_k as the resulting quantity and time from price calculation for resource k. We do not take the request parameters as filter criteria for tariffs, but as a minimum requirement. We take this approach because we do not know ex ante if a tariff with more resources or a longer period is more expensive or even cheaper (e.g. if a request states 11 months of run-time, a 1-year reservation-based tariff probably is favorable). Therefore, we compute all tariffs that fulfill the minimum requirements and then compare total costs of all tariffs.

A consumer might not utilize resources for the full duration of T. After all, the flexibility and scalability of resource usage *and* costs is a major advantage of cloud infrastructure which we may not neglect. Therefore, we define a target utilization rate κ with $0 < \kappa \leq 1$ which models the share of T when the resources are actually in use. For example, $\kappa = 0.75$ means that a VM is online 75% of the time or 18 hours a day. Initially, for each single price, we set $q_k = Q_k$ and $t_k = \kappa T$ with a default value of $\kappa = 1$. In addition, for proper discount determination, we need to know if the user is willing to pay in advance for a tariff. If so, the parameter θ is set to 1 which is the default.

5.2.2. Pricing

Table 3 contains the attributes of the `Price` entity which models the data and conditions necessary for price calculation. Besides the price itself (`netamount`) in a defined `currency`, prices have two dimensions as denominators – `quantity` and `time` (e.g. *gigabyte* per *month* for an amount of traffic in a specific period). One of them or both can be missing, e.g. if `quantity` is missing, we only have "EUR/hour" or for a one-time fixed fee, both `quantity` and `time` would be missing and we would only have "EUR". To keep

Table 2: Request parameters for price calculation

Parameter	**Description**
Q_k	Requested quantity of a resource k
q_k	Effective quantity of resource k from price calculation
t_k	Effective time period of resource k from price calculation
T	Requested usage period for the deployment
κ	Planned utilization of resources during T
θ	Equals 1 (*true*) if user accepts prepayment (default)

Table 3: `Price` properties

Field	**Description**	**Symbol**
name	Display name of the price (e.g. "Base fee")	
netamount	The net value for this price	
currency	Currency of price (ISO 4217 code)	
quantity	Quantity denomination for this price	
time	Time denomination for this price	
validityPeriod	Period for which this price is valid	f_v
applicabilityRange	The quantity range for which this price is applicable	f_a
partitionRange	Quantity share for which this price is valid	q
bookingPeriod	Required minimum booking period for this price (e.g. 1 year)	t
minimumCharge	Indicates if this price nominates a minimum charge	p_i

price modeling flexible for different quantity units, we introduce a type parameter for the quantity unit to the Price type so it can be instantiated with any necessary quantity unit (using the auxiliary types for quantity we defined in Appendix A.1).

Equation 8 shows the computation of a single price p_i from its components. Every single price p_i of a `TariffItem` is evaluated on its own (Equation 8) and afterwards all results are aggregated to the total `TariffItem` cost c_j (Equation 9) Besides `quantity` and `time`, `Price` also contains the conditions we mentioned in Section 4.2 which we formalize below.

`applicabilityRange` The filter flag f_a indicates if the price p_i is valid for the quantity Q_k. If `applicabilityRange` is set (checked by $\exists$ function) and the quantity Q_k is outside of the bounds, f_a switches to 0 and thus p_i is 0 as well (cf. Equation 8).

$$\begin{aligned} f_a = &(1 - \exists\texttt{applicabilityRange}) + \\ &(\underline{\texttt{applicabilityRange}} < Q_k) * (Q_k \leq \overline{\texttt{applicabilityRange}}) \end{aligned} \tag{4}$$

`partitionRange` For partitioned pricing, this attribute contains a quantity range for which the current price p_i is valid. In contrast to `applicabilityRange` which denotes the price for the *total quantity*, `partitionRange` indicates a price for the *share of* the quantity within this range. The usage of fields `applicabilityRange` and `partitionRange` is mutually exclusive.

If the current price p_i is a partitioned price, the calculation quantity q_k defines the quantity share for the current price p_i. First, we check for existence of a partitioned price. If it is not defined, we stick with the total purchase quantity Q_k. Otherwise, we compute the share of Q_k to apply for the current price with the second term.

$$\begin{aligned} q_k =& (1 - \exists \texttt{partitionRange}) * Q_k + \\ & |\max(Q_k - \underline{\texttt{partitionRange}}, 0) - \max(Q_k - \overline{\texttt{partitionRange}}, 0)| \end{aligned} \tag{5}$$

We distinguish three cases which are captured by Equation 5:

1. $Q_k < \underline{\texttt{partitionRange}} < \overline{\texttt{partitionRange}}$: The purchase quantity is below the partition range which means the first parameter of both max functions is < 0 and the quantity share q_k results to 0.
2. $\underline{\texttt{partitionRange}} < Q_k < \overline{\texttt{partitionRange}}$: The purchase quantity lies within the range, but not above the upper limit and the resulting quantity share within the range is $q_k - \underline{\texttt{partitionRange}}$.
3. $\overline{\texttt{partitionRange}} < Q_k$: The purchase quantity is above the range, so the full quantity of $\overline{\texttt{partitionRange}} - \underline{\texttt{partitionRange}}$ needs to be applied to the price.

`bookingPeriod` This field indicates the minimum time period which applies for the price. We consider this period when computing cost.

To determine the correct calculation period t for the price p_i, we use either the target period κT or the `bookingPeriod` opposed by the price conditions – whichever is larger (Equation 6). If a period from previously processed prices has already imposed a longer period, we keep the longer period of all periods ($max(t_*)$) as we assume that all resources of a request are consumed for the same period.

$$t = \max(\kappa T, \texttt{bookingPeriod}, max(t_*)) \tag{6}$$

`validityPeriod` We keep track in which calender period a price has been valid and only currently valid prices will be considered in cost computation. Quite frequently, providers lower prices, but keep the technical offer as is which means we just add prices with a new validity period and let the old ones expire. As a side effect, we keep track of a pricing history which is useful for e.g. trend or industry analyses.

The filter flag f_v equals 1 if the price p_i is currently valid given its `validityPeriod`. Otherwise, it is 0 which removes outdated prices. *now* returns the current time.

$$f_v = (\underline{\texttt{validityPeriod}} \leq \text{now}) * (\text{now} \leq \overline{\texttt{validityPeriod}}) \tag{7}$$

Table 4: `Discount` properties

Field	Description
name	Display name of the discount (e.g. "Prepay discount")
discountFactor	Factor to apply on cost when discount is applicable
scope	Applicable level (Resource vs. Tariff)
utilizationRange	Target Utilization to apply discount
spendRange	Target spend range to apply discount
prepay	Discount applies when cost are paid in advance
committedPeriod	Minimum booking period to apply discount

`minimumCharge` This is a boolean indicator classifying the current price as a minimum consumption amount. For price calculation, it means that the resulting cost for the tariff will always be at least as high as the `netamount` of the price flagged as minimum charge price, which we denote as $p_{\text{minimumCharge}}$. If the computed tariff cost C_{T+} is above $p_{\text{minimumCharge}}$, the latter has no further effect.

Finally, Equation 8 shows the actual single price computation applying the determined quantity and time and the control flags as defined above. If the price is not valid, not applicable for the quantity or a minimum charge, $p_i = 0$. Otherwise, the determined quantity q and t are normalized with the denominating quantity and time of the price (e.g. 10 GB per 1 GB or 24h per 1 month) to apply the right factor on `netprice`. The factor $(1 - \texttt{minimumCharge})$ ensures that a price flagged as minimum charge will not be added to the total item costs c_j, but used later for comparison in Equation 9.

$$p_i = f_v * f_a * (1 - \texttt{minimumCharge}) * \texttt{netprice} * \frac{q}{\texttt{quantity}} * \frac{t}{\texttt{time}} \tag{8}$$

The resulting cost c_j for the `TariffItem` now is either the sum of all single prices p_i or $p_{\text{minimumCharge}}$, whichever is higher:

$$c_j = \max(\sum_{i=1}^{n} p_i, p_{\text{minimumCharge}}) \tag{9}$$

5.2.3. Discount Modeling

We apply discounts in a separate step after all costs have been calculated to decrease complexity and because the total cost influences the achievable discounts. At the core, there is a `discountFactor` stating the percentage decrease of a cost base if the discount applies. Note that also negative `discountFactorss` are possible in case *surcharges* needs to be modeled.

`Discounts` have a scope stating to which cost aggregate they apply. We distinguish *Resource* and *Tariff* scope which signal to apply `discountFactor` either to the `TariffItem` costs c_j or to the total cost of the `Tariff`, C_{T+}, respectively. We define γ as the relevant cost base for discounts as

$$\gamma = \begin{cases} c_j, & \text{if } \texttt{scope} = \text{RESOURCE} \\ C_{T+}, & \text{if } \texttt{scope} = \text{TARIFF} \end{cases} \tag{10}$$

We distinguish the following discount conditions represented by the defined control variables δ_x. If a condition is not specified (checked by the $\exists$ function), the value of the control variable switches to 1 because a restriction which is not set, is fulfilled:

Spend Range If providers grant a discount when a consumer reaches certain spend levels within a month, the `discountFactor` is applicable if the relevant spend γ is within the range:

$$\begin{aligned} \delta_s =& (1 - \exists \texttt{spendRange}) + \\ & (\underline{\texttt{spendRange}} < \gamma) * (\gamma \leq \overline{\texttt{spendRange}}) \end{aligned} \tag{11}$$

Period Commitment We consider a discount on a commitment for a certain time period as applicable when the period commitment is not longer than the purchase period T (see 5.2.1):

$$\delta_t = (1 - \exists \texttt{committedPeriod}) + (\texttt{committedPeriod} \leq T) \tag{12}$$

Utilization Range The `discountFactor` applies if the user reaches a certain range of monthly utilization. To control this discount, we define a rule against the user's target utilization κ:

$$\begin{aligned} \delta_u =& (1 - \exists \texttt{utilizationRange}) + \\ & (\underline{\texttt{utilizationRange}} < \kappa) * (\kappa \leq \overline{\texttt{utilizationRange}}) \end{aligned} \tag{13}$$

Prepay We introduce a control flag δ_p which switches to 0 if and only if the discount requires prepayment ($\texttt{prepay} = 1$), but the user denied to pay in advance in the request ($\theta = 0$, see 5.2.1):

$$\delta_p = 1 - \texttt{prepay} * (1 - \theta) \tag{14}$$

The smallest scope level to apply discounts is on resource level, i.e. based on the cost of a `TariffItem` (cf. Figure 4). To achieve the resulting monetary value d_j for a `TariffItem` with h discounts attached, we calculate:

$$d_j = \sum_{i=1}^{h} (\delta_s * \delta_t * \delta_u * \delta_p * \texttt{discountFactor} * \gamma) \tag{15}$$

Table 5: Sample tariff 1 to model with the OTM

Tariff Component	**Configuration**
Name	m3.large, Reserved 1 year, Partial Upfront
Datacenter	Frankfurt, Germany
VM	2 CPUs, 7.5 gigabytes, 32 GB SSD Storage
Price	Upfront Payment: $492, Hourly Rate: $0.054

If all defined conditions are met, the latter term in Equation 15 applies the given `discountFactor` to the appropriate cost base γ. Equation 3 shows the aggregation of discounts to the total reduction D which we apply in Equation 1 to arrive at net cost C_T (after discounts) for this tariff.

Please note that we are able to add further discount conditions by defining appropriate control flags and extend the discount computation. Also additional scope levels for discounts such as *Provider*- or *Data Center*-level are easy to extend.

6. Model Application

We chose a tariff to demonstrate the application of the OTM on a real-world example and how to map the different properties into the OTM structure. We use Java-style code samples to provide a transparent demonstration of the model application. Along with our explanation, we show only relevant code samples while the full listing for the tariff example is in Appendix B (we provide line numbers for reference). For a real-world application, one would create a tariff editor supporting batch definition of tariffs to make data collection more efficient. Also, a user interface to specify requests against the tariff data base and browse results would be necessary for an end-user application.

As the market leader, AWS plays a reference role in the IaaS market and hence we want to demonstrate how a tariff of this provider would appear in the OTM. We chose the *m3.large* tariff [Amazon Web Services, 2015] in its 1-year reservation-based variant. If the `Provider` and the `DcLocation` should not exist in the database, we instantiate both and assign the data center to the provider (App. B, 2-4). Other properties to set in a real-world case would be geographic location or address of the DC (*Frankfurt, Germany* in this case) and contact information for the provider.

6.1. Tariff modeling

6.1.1. Resources

Before we create the actual `Tariff`, we define all the resource that are used together with `Price`s to build `TariffItem`s. The `Tariff` then holds a list of `TariffItem`s which belong together as defined by the provider's offer.

VM Creating the `VmResource` for this tariff is straight-forward – we just need to create the instance and set the properties applying the correct `QuantityUnits` (see App. A.1).

```
8   VmResource vm = new VmResource();
9   vm.setCpuCores(new CpuCore(2));
10  vm.setRamResource(new GB(7.5));
```

Storage Regarding the `StorageResource`, we need to flag it as internal VM storage, set the size and register it as an SSD drive. The internal storage has a fixed size of 32 GB, hence we set it as the instance size. As there is no choice about the size of the drive, the `validValue` property is also set to the actual size. If there would be choices, we could add them to the list of valid values to indicate this choice.

```
13  StorageResource storage = new StorageResource();
14  storage.setSubtype(StorageResource.INTERNAL);
15  storage.setSize(new GB(32));
16  storage.setSsd(true);
17
18  storage.setValidValues(new ValueRange(new GB(32)));
```

Traffic As stated in section 5.1, we only distinguish between incoming and outgoing traffic, indicated by the marker values `TrafficResource.INGRESS` and `TrafficResource.EGRESS`. AWS does not impose a restriction on the amount traffic, so we do not need to specify any `validValues`.

```
21  TrafficResource inTraffic = new TrafficResource();
22  inTraffic.setDirection(TrafficResource.INGRESS);
23
24  TrafficResource outTraffic = new TrafficResource();
25  outTraffic.setDirection(TrafficResource.EGRESS);
```

Operating System, NetworkResource AWS offers many different operating system images for which we would need to create separate resource instances each. We apply a default Linux OS for this example and register it as an Linux OS by setting the `family` to `OsResource.LINUX` (App. B, 28-31). For a network component, we model the option of assigning a static IP address to the VM as a `NetworkResource` (App. B, 34-35). Again, there is no explicit restriction on the number of IP addresses, thus no further quantity restrictions need to be specified.

6.1.2. Pricing

Now that the technical resources have been defined, we model the commercial part of the tariff. To do so, we create a `TariffItem` which assigns one or more prices and discounts (if applicable) to a resource. The collection of all items builds up the `Tariff`.

The reservation-based tariff runs for one year and consists of a partial upfront payment and a reduced hourly rate (compared to the on-demand tariff). We create the `Tariff`

and assign it to the respective `DcLocation` and `Provider` (App. B, 39-41). After creating a `TariffItem`, we set the corresponding properties and assign the `VmResource` and the `Price`s to the `TariffItem` which we add to the `Tariff`.

As the `StorageResource` is included in the VM price, we give it a price of 0 with the same booking period as the VM (App. B, 66-75).

```
44    TariffItem vmItem = new TariffItem();
45    vmItem.setResource(vm);
46
47    Price upfront = new Price("Upfront payment");
48    upfront.setCurrency("USD");
49    upfront.setNetAmount(492);
50    upfront.setBookingPeriod(new Year(1));
51
52    vmItem.addPrice(upfront);
53
54    Price hourly = new Price("Hourly rate");
55    hourly.setCurrency("USD");
56    hourly.setNetAmount(0.054)
57    hourly.setPerQuantity(new Count(1));
58    hourly.setPerTime(new Hour(1));
59    hourly.setBookingPeriod(new Year(1));
60
61    vmItem.addPrice(hourly);
62
63    tariff.addItem(vmItem);
```

For `TrafficResource`, we show an example of quantity dependent pricing. We use the `partitionRange` attribute of `Price` to assign the correct price for each quantity of traffic. If the overall traffic amount per month would determine the price for *all* the traffic, we would need to use `applicabilityRange` instead. To specify the valid quantity range for this price we utilize the auxiliary entity `ValueRange` (App. A.2). Also we apply the `DataUnit` hierarchy (App. A.1) which allows us to mix units like `GB` and `TB` (as values are noted in the provider's tariff list) but ensures that correct conversion will take place. As all ingress is free, we set its price to 0 (App. B, 81-86).

```
89    TariffItem egressItem = new TariffItem("Egress");
90    egressItem.setResource(outTraffic);
91
92    // Egress differs per quantity
93    Price firstgb = new Price("Egress, First GB");
94    firstgb.setCurency("USD");
95    firstgb.setNetAmount(0);
96    firstgb.setPerQuantity(new GB(1));
97    firstgb.setPerTime(new Month(1));
98    firstgb.setPartitionRange(new ValueRange<DataUnit>(new GB(0), new GB
          (1)));
99    egressItem.addPrice(firstgb);
100
101   Price upTo10Tb = new Price("Egress, 1GB-10TB");
102   upTo10Tb.setCurrency("USD");
```

Table 6: Reserved Instance Volume Discounts of AWS

Reserved Instance Turnover	Discount Rate
<$500,000	0%
$500,000 - $4,000,000	5%
$4,000,000 - $10,000,000	10%

```
103     upTo10Tb.setNetAmount(0.09);
104     upTo10Tb.setPerQuantity(new GB(1));
105     upTo10Tb.setPerMonth(new Month(1));
106     upTo10Tb.setPartitionRange(new ValueRange<DataUnit>(new GB(1), new TB
            (10)));
107     egressItem.addPrice(upTo10Tb);
```

For further resources like OS and Static IP, we apply similar commands to complete the pricing model of the tariff (App. B, 120-137).

6.1.3. Discounts

Regarding discounts, we like to model the *Reserved Instance Volume Discounts* (see Table 6). This is a simple volume discount on turnover. The first line of Table 6 has no effect as the discount rate is 0%, so we model the first discount for the second line (Table 6). As this discount applies only to the reserved instance turnover, we set its scope to *RESOURCE* (cf. 5.2.3) and then add it to the VM tariff item. The creation of the last discount stage in Table 6 is identical, but with adjusted values (App. B, 147-152).

```
140  Discount discount1 = new Discount("500k-4000k");
141  discount1.setSpendRange(new ValueRange<Double>(500000,4000000));
142  discount1.setDiscountFactor(0.05);
143  discount1.setScope(Discount.RESOURCE_LEVEL)
144
145  vmItem.addDiscount(discount1);
```

Now as we have created the tariff structure in the OTM, we are able to use it for cost estimates. For other tariffs, the next steps works identically, applying the semantics of the model and hence enabling a uniform cost estimation of all tariffs once they have gone to the above steps of mapping their resource and pricing into the OTM.

6.2. Sample calculation

To show how cost estimation works, we assume a simple request (Table 7). All values not specified in the request stay at their default and do not serve as restrictions, e.g. it does not matter if the tariff has a `NetworkResource` or if it offers a prepaid option or not (θ missing).

The first step is the matching of requested resources against the tariff. As we can easily see, the tariff's VM size is sufficient for the requested `VmResource` (because it has

Table 7: Sample Request to AWS tariff

Resource/Parameter	**Value**
VmResource	CPU=CpuCores(2), RAM=GB(6)
StorageResource	GB(20)
Outgoing Traffic	GB(100)/Month(1)
T	Month(10)
κ	1

at least 2 CPU cores and 6 GB RAM), dito for the `StorageResource`. As the tariff does not restrict traffic volume, this is a match as well.

6.2.1. Cost estimation

For price calculation, we start with the `VmResource` and apply Equation 8 to the first price (*upfront* in 6.1.2). We assume the price is currently valid ($f_v = 1$) and no `applicabilityRange` is set ($f_a = 1$). The price is also no minimum charge, so we focus on the quantity and time factors. We requested one VM and thus $q = Q = \texttt{quantity} = \texttt{Count(1)}$. For t, we apply Equation 6 which results in $t = \max(1 * \texttt{Month(10)}, \texttt{Year(1)})$ and hence $t = \texttt{Year(1)}$. Applied to Equation 8, we achieve

$$p_1 = 1 * 1 * 1 * 492 * \frac{1}{1} * \frac{\texttt{Year(1)}}{\texttt{Year(1)}} = 492$$

The second price (*hourly*) p_2 of `VmResource`, when inserted in Equation 8:

$$p_2 = 1 * 1 * 1 * 0.054 * \frac{1}{1} * \frac{\texttt{Year(1)}}{Hour(1)} = 473.04$$

p_2 is the last price for the VM `TariffItem` so we aggregate the price positions to the first `TariffItem`'s total cost:

$$c_1 = p_1 + p_2 = 492 + 473.04 = 965.04$$

As mentioned above (6.1.2), the `StorageResource` has a `netamount` of 0, so its only price $p_1 = 0$ and $c_2 = 0$. For traffic, we need to apply partitioned pricing which we start with the first price. Before, we need to determine the partition quantity by Equation 5:

$$\begin{aligned} q_1 &= |\max(\texttt{GB(100)} - \texttt{GB(0)}, 0) - \max(\texttt{GB(100)} - \texttt{GB(1)}, 0)| \\ &= |\texttt{GB(100)} - \texttt{GB(99)}| = \texttt{GB(1)} \end{aligned}$$

When we apply Equation 8 to this quantity for the first price (*firstgb*) of `TariffItem` 3, we arrive at $p_1 = 0$, as the `netamount` of this price is 0. For the second price partition, we use again Equation 5 for the partition quantity. Because the first parameter of the second max function is negative, it evaluates to 0 (the second parameter):

$$
\begin{aligned}
q_2 &= |\max(\texttt{GB(100)} - \texttt{GB(1)}, 0) - \max(\texttt{GB(100)} - \texttt{TB(10)}, 0)| \\
&= |\texttt{GB(99)} - 0| = \texttt{GB(99)}
\end{aligned}
$$

We use this result to compute the price p_2 using Equation 8 with the period of one year determined by the run of `VmResource`:

$$
\begin{aligned}
p_2 &= 1 * 1 * 1 * 0.09 * \frac{\texttt{GB(99)}}{\texttt{GB(1)}} * \frac{\texttt{Year(1)}}{\texttt{Month(1)}} \\
&= 0.09 * 99 * 12 = 106.92
\end{aligned}
$$

Again, we arrived at the last price for this `TariffItem` and thus we total the cost:

$$c_2 = p_1 + p_2 = 0.00 + 106.92 = 106.92$$

Now we processed all `TariffItem`s and use Equation 2 to compute total tariff costs before deduction of discounts C_{T+} which closes the pricing computation:

$$C_{T+} = c_1 + c_2 + c_3 = 965.04 + 0.00 + 106.92 = 1071.96$$

6.2.2. Discounting

After we finished the price computation, we examine applicable discounts. We have two discounts on *RESOURCE* level for the VM `TariffItem`, hence the discount base $\gamma = c_1 = 965.04$.

Using Equation 15, we determine the absolute rebate after determining the discount flags δ_x. As we only specified `spendRange`, $\delta_t = \delta_u = \delta_p = 1$ by definition. For δ_s, we get

$$\delta_s = (500,000 < 965.04) * (965.04 \leq 4,000,000) = 0$$

which results in $d_1 = 0$ for Discount 1 using Equation 15. Likewise, we arrive at $d_2 = 0$ for the second discount which is, obviously, the correct result as the spend of 965.04 is below the minimum of 500,000. As these are all modeled discounts, we arrive at a total discount value of $D = 0$, i.e. none of the discounts are applicable. Finally, the net tariff cost for the given sample request upon the tariff example results to $C_T = C_{T+} - D = C_{T+} = 1071.96$.

This was a simplified sample case for a tariff which is easy to compute manually. But with growing number of providers, datacenters, tariffs, resources, options and prices, the computation becomes tedious and more complicated. Here, a decision support system implementing the OTM model can help to compare offers more efficiently and arrive at more precise cost estimates for infrastructure setups.

7. Conclusion

In this paper, we proposed a flexible and powerful description model for cloud service offers. It provides a lightweight structure with only a few entities that serve as building blocks for complex tariff structures. It is also extensible either by adding additional properties to the resource types or by introducing new entities for future types of cloud services. Auxiliary types, such as a generic modeling of quantity and time provide an abstract interface for resource and tariff processing and ensure consistent conversion and computation.

With our work, it is now possible to collect heterogeneous tariff data in a common model for decision processes or other analytical purposes. This has an enabling effect for future work in this area. For researchers, an extensive database of detailed market data based on the OTM provides a rich foundation for insights into market development, segmentation, provider behavior, technology leadership, success of product portfolios etc. In addition, the development of rich decision support technologies based on the OTM as a language to model this complex market and provide the necessary data to feed these systems.

Applications for practitioners include sourcing decision support such as vendor comparison or make-or-buy analyses. To gain deeper insights into the cloud infrastructure market, the OTM provides a database to analyze market segmentation and vendor portfolios, pricing politics and price development. Specifically, for cloud providers, it provides data for product and price comparisons, e.g. by computing prices of different standard portfolios and analyze the effects of the own pricing scheme.

The OTM is also suitable for broker systems which support sourcing decisions (e.g. [Cunha et al., 2013, Gottschlich et al., 2014]) by automating request for proposal (RFP) processes. The resources hierarchy serves to formulate the required demands which the system processes for every provider tariff to get a cost estimate. By using the same resource types for querying and storing the data, we can encapsulate the matching logic with the specific resource type (i.e. a `VmResource` decides if it matches another `VmResource`) and keep the processing logic abstract on `GenericResource` level. Thus, we contain the extensibility property of the OTM, but do not need to adjust the matching logic when new resources are introduced. The pricing logic is also generic for the involved quantities and resources and hence does not hinder resource extensions.

As performance of cloud services differ, for a comprehensive decision for a certain product, the price comparison provided by our model needs to be complemented by benchmarks to ensure a proper performance comparison. For IaaS, we can use benchmark tools for CPUs, storage etc. to derive performance metrics (cf. e.g. [Gottschlich et al., 2014]). Those metrics could then be included in the model to query for certain minimum requirements in terms of performance. For suitable performance-related service comparison tests see e.g. [Li et al., 2010, Silva et al., 2013, Cunha et al., 2013, Gillam et al., 2013].

The proposed model and the method of its creation also serve as an example or template for other markets with complex products, e.g. RFPs for buildings in the construction sector. There, we also have many single tasks which might differ in details

and quality (e.g. which materials to use for thermal isolation) and include certain price components or not. In the end, a meaningful combination of the elementary tasks and a proper aggregation of costs is necessary to arrive at a completed building – just like a working IT infrastructure requires many components to work together properly.

References

[Akerlof, 1970] Akerlof, George a 1970. The Market for "Lemons": Quality Uncertainty and the Market Mechanism. The Quarterly Journal of Economics, 84(3):488–500.

[Amato & Venticinque, 2013] Amato, Alba, & Salvatore Venticinque 2013. Multi-objective decision support for brokering of cloud SLA. In Proceedings - 27th International Conference on Advanced Information Networking and Applications Workshops, WAINA 2013, pages 1241–1246. IEEE.

[Amazon Web Services, 2015] Amazon Web Services 2015. AWS — Amazon EC2 — Pricing. In https://aws.amazon.com/ec2/pricing/.

[Centron GmbH, 2015] Centron GmbH 2015. Preise :: Cloud Hosting ccloud Preise. In https://www.centron.de/produkte/cloud-hosting/ccloud/preise.html.

[Cunha et al., 2013] Cunha, Matheus, N Mendonça, & A Sampaio 2013. A Declarative Environment for Automatic Performance Evaluation in IaaS Clouds. In Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing, pages 285–292.

[DMTF, 2013] DMTF 2013. Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP-based Protocol - An Interface for Managing Cloud Infrastructure. Technical report, Distributed Management Task Force, Document DSP0263, Version 1.1.0.

[El Kihal et al., 2012] El Kihal, Siham, Christian Schlereth, & Bernd Skiera 2012. Price comparison for infrastructure-as-a-service. In ECIS 2012 Proceedings, pages 1–12, Paper 161.

[Garg et al., 2011] Garg, Saurabh Kumar, Steve Versteeg, & Rajkumar Buyya 2011. SMICloud: A Framework for Comparing and Ranking Cloud Services. In 2011 Fourth IEEE International Conference on Utility and Cloud Computing, number Vm, pages 210–218. Ieee.

[Garg et al., 2013] Garg, Saurabh Kumar, Steve Versteeg, & Rajkumar Buyya 2013. A framework for ranking of cloud computing services. Future Generation Computer Systems, 29(4):1012–1023.

[Gillam et al., 2013] Gillam, Lee, Bin Li, John O'Loughlin, & Anuz Pratap Tomar 2013. Fair Benchmarking for Cloud Computing systems. Journal of Cloud Computing: Advances, Systems and Applications, 2(1):6.

[Glaser & Strauss, 2009] Glaser, Barney G, & Anselm L Strauss 2009. The discovery of grounded theory: Strategies for qualitative research. Transaction Publishers.

[Google, 2015] Google 2015. Google Compute Engine Pricing - Compute Engine. In https://cloud.google.com/compute/pricing.

[Gottschlich et al., 2014] Gottschlich, Jörg, Johannes Hiemer, & Oliver Hinz 2014. A Cloud Computing Broker Model for IaaS Resources. In ECIS 2014 Proceedings, pages 1–15, Track 10, Paper 8.

[Hevner et al., 2004] Hevner, Alan R., Salvatore T. March, Jinsoo Park, & Sudha Ram 2004. Design science in information systems research. Mis Quarterly, 28(1):75–105.

[Li et al., 2010] Li, Ang, Xiaowei Yang, Srikanth Kandula, & Ming Zhang 2010. CloudCmp: Comparing Public Cloud Providers. In Proceedings of the 10th annual conference on Internet measurement - IMC '10, pages 1–14, New York, New York, USA. ACM Press.

[Menzel et al., 2013] Menzel, Michael, Marten Schönherr, & Stefan Tai 2013. (MC2)2 : criteria, requirements and a software prototype for Cloud infrastructure decisions. Software: Practice and Experience, 43(11):1283–1297.

[Metsch & Edmonds, 2011] Metsch, T, & Andy Edmonds 2011. Open Cloud Computing Interface - Infrastructure. Technical report, Open Grid Forum, GFD-P-R.184,, Ver. 1.1.

[Moore, 2015] Moore, Susan (Gartner) 2015. Gartner Says Worldwide Cloud Infrastructure-as-a-Service Spending to Grow 32.8 Percent in 2015. In http://www.gartner.com/newsroom/id/3055225.

[Nobel, 2011] Nobel, Rickard 2011. Storage performance: IOPS, latency and throughput. In http://rickardnobel.se/storage-performance-iops-latency-throughput/.

[Patiniotakis et al., 2014] Patiniotakis, Ioannis, Yiannis Verginadis, & Gregoris Mentzas 2014. Preference-based cloud service recommendation as a brokerage service. In Proceedings of the 2nd International Workshop on CrossCloud Systems - CCB '14, pages 1–6, Paper 5.

[Preimesberger, 2015] Preimesberger, Chris 2015. Big Four IaaS Providers Now Own Half the Market. In Eweek, http://www.eweek.com/cloud/big-four-iaas-providers.

[ProfitBricks, 2015] ProfitBricks 2015. Cloud Pricing - Save on Cloud Server Pricing — ProfitBricks.

[Rehman et al., 2011] Rehman, Zia Ur, Farookh K. Hussain, & Omar K. Hussain 2011. Towards Multi-criteria Cloud Service Selection. 2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, pages 44–48.

[Repschlaeger et al., 2013] Repschlaeger, Jonas, S Wind, R Zarnekow, & Klaus Turowski 2013. Decision Model for Selecting a Cloud Provider: A Study of Service Model Decision Priorities. In AMCIS 2013 Proceedings, pages 1–11, Paper 27.

[Roovers et al., 2012] Roovers, Joris, Kurt Vanmechelen, & Jan Broeckhove 2012. A reverse auction market for cloud resources. In Economics of Grids, Clouds, Systems, and Services - 8th International Workshop, GECON 2011, Paphos, Cyprus, December 5, 2011, Revised Selected Papers, pages 32–45.

[Siebenhaar et al., 2011] Siebenhaar, Melanie, Ulrich Lampe, Tim Lehrig, Z Sebastian, Stefan Schulte, & Ralf Steinmetz 2011. Complex Service Provisioning in Collaborative Cloud Markets. In Abramowicz, Witold, Ignacio M. Llorente, Mike Surridge, Andrea Zisman, & Julien Vayssière (eds), Towards a Service-Based Internet - 4th European Conference, ServiceWave 2011, Poznan, Poland, October 26-28, 2011. Proceedings, volume 6994 of *Lecture Notes in Computer Science*, pages 88–99. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Silva et al., 2013] Silva, Marcio, Michael R. Hines, Diego Gallo, Qi Liu, Kyung Dong Ryu, & Dilma Da Silva 2013. CloudBench: Experiment Automation for Cloud Environments. 2013 IEEE International Conference on Cloud Engineering (IC2E), pages 302–311.

[Simon, 1996] Simon, H. A. 1996. The Sciences of the Artificial. MIT Press, Cambridge, MA, 3rd editio edition.

[The Cloud Service Measurement Initiative Consortium (CSMIC), 2011] The Cloud Service Measurement Initiative Consortium (CSMIC) 2011. Service Measurement Index Introducing the Service Measurement Index (SMI).

[Wang et al., 2013] Wang, Wei, Di Niu, Baochun Li, & Ben Liang 2013. Dynamic Cloud Resource Reservation via Cloud Brokerage. In 2013 IEEE 33rd International Conference on Distributed Computing Systems, pages 400–409. IEEE.

A. Auxiliary Entities

This section introduces some auxiliary entities for quantity and time modeling which are necessary to connect resource consumption and price application. We introduce these entities to enable an abstract quantity and time specification of resources and prices. This enables us to keep the price computation generic and the model extensible for new cloud service types which might bring their own units of measurement.

A.1. Quantity Modeling

We need to specify resource quantity in at least four contexts:

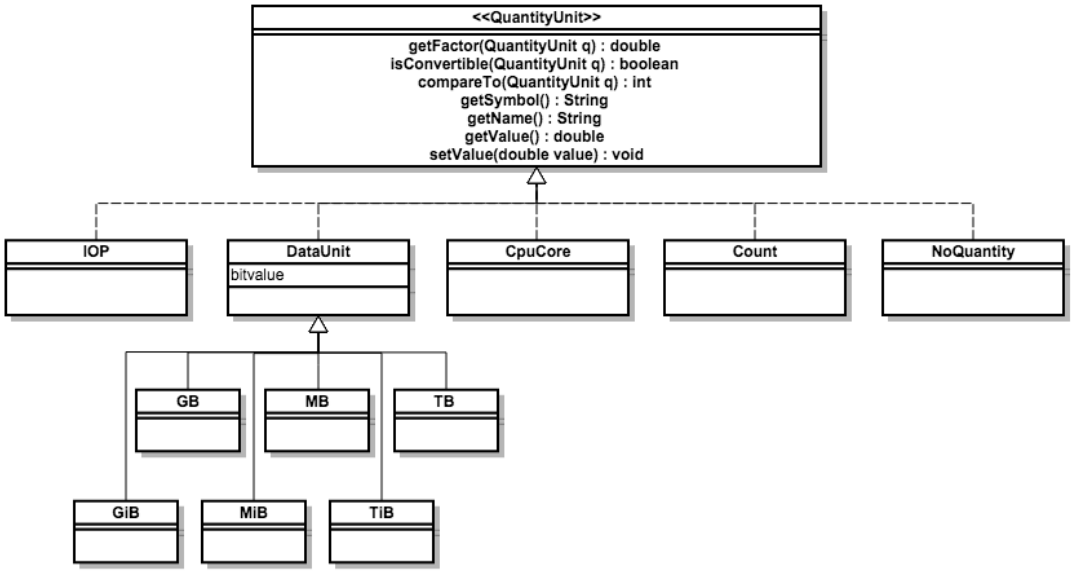

Figure 5: Auxiliary Quantity Model for OTM

1. To specify the granularity a provider offers (e.g. CPUs are available with 1, 2, 4, 8 and 16 cores). Those unit sizes for resource are either discrete (for cpu cores) or continuous (e.g. capacity in gigabytes for storage or traffic).

2. For a particular resource instance, we need to store the actual capacity or size. For example, if a VM contains 20 gigabytes internal storage, we need to store that exact amount with the storage resource.

3. Prices usually apply to a certain quantity or capacity (e.g. GB) which needs to be specified in a way compatible to resource quantity so that prices can be applied to resources

4. When a specific request is made, the resource capacity desired by the user needs to be quantified and stored with the resource.

The challenge for a common pricing model is to find a generic approach for different units of measurement (e.g. 1000 GB traffic volume for `TrafficResource` or [2 cores; 8 GB RAM] for `VmResource`) which is also extendable as the model itself and still allows for a generic processing of the different resource types and their prices.

Figure 5 shows the quantity model for the currently defined resource types (cf. 5.1). All quantity types implement a common interface which provides the functionality to generically handle quantity processing to match a request for resources and compute the costs. Despite some identifying functionality (name, symbol etc.) this behavior includes:

Conversion Using the `getFactor` function with a convertible `QuantityUnit`, the inter-

face returns the appropriate conversion factor or *null* if the units are not convertible.

Convertability The `isConvertible` function returns *true* if the given `QuantityUnit` is convertible with the current one

Comparison The `compareTo` function implements an ordering function which checks if the value of a given QuantityUnit instance is higher, lower or equal. Using this helps e.g. to decide whether a certain resource configuration satisfies a requested resource demand (cf. App. A.2).

`DataUnit` is the most interesting implementation of a `QuantityUnit` as it allows to express storage sizes in any of the magnitudes commonly used (e.g. `new GB(500);`), but ensures at the same time that they are correctly converted and compared to each other. At the same time, we handle the issue that some providers use a factor of 1000 between mega-, giga- and terabyte and others use 1024 (2^{10}), also known as mebi-, gibi- and tebibyte. By using the right unit on data entry, the implementation ensures correct handling of sizes. `CpuCore` is a unit of measurement to label the count of CPU cores correctly, e.g. when describing a VM configuration (cf. 6).

Some providers have prices based on `IOPs` (Input/Output operations), so we include an appropriate `QuantityUnit`. `Count` serves as quantity for all countable resources (e.g. static IP addresses). `NoQuantity` finally is a placeholder in case a `Price` has explicitly no quantity unit defined (e.g. a simple monthly fee).

Besides using the `QuanityUnit` interface as data type to specify quantities within the resources, we also use it as a type parameter for `Price` to express the unit of measurement for the price – no matter how it is structured. Thus, the OTM is able to express resource and billing quantities for future resource types as soon as they are defined. In addition to the abstraction capabilities of the `QuantityUnit`, it also contribute to type safety because they ensure we avoid inconsistencies in pricing (e.g. a price per IOP for a `RamResource`).

A.2. ValueRange – specifying resource quantity ranges

The resource instances for a certain provider tariff contain a list of valid quantities, e.g. for `CpuResource` it contains the list of the core quantities the provider offers, e.g. 2, 4, 8 or 16 cores. We need to match a resource demand against this list of values to arrive at a valid granularity of resource configuration for a provider. A demand of 6 cores would be satisfied by an 8-core machine if the provider does not offer 6-core machines. Other resources might not have discrete values, but a continuous range of possible resource configurations (e.g. storage is available in any amount between 1 and 1000 gigabytes).

To model these configurations, we introduce a data type `ValueRange` which tracks the valid resource configurations for resources (cf. `validValues` in 5.1) and supports both a discrete list as well as a continuous range of quantity. It takes a `QuantityUnit` as a type parameter and is able to deliver a valid resource configuration when queried with a resource demand. We use the functionality provided by the `QuantityUnit` interface

to manage this list (e.g. `compareTo` for ordering). `ValueRange` offers the following functionality:

isWithin(QuantityUnit q) Returns true if value is present in the list

searchEqualOrHigher(QuantityUnit q) Returns the value of the list which is either equal to parameter value or the next higher value from the list

searchEqualOrLower(QuantityUnit q) Returns the value of the list which is either equal to parameter value or the next lower value from the list

searchClosest(QuantityUnit q) Returns either the same value if is in the list or the closest next one, be it higher or lower

If the valid values for a `GenericResource` descendant is not defined, then there are no restrictions (known) on the possible quantity amounts and the resource is available in any quantity. We also use `ValueRange` to specify quantity ranges in `Price`s, e.g. `applicabilityRange` (see 6). The functionality of `ValueRange` helps to efficiently and generically check price configurations during cost calculation.

A.3. Time Modeling

Similar to the quantity modeling (App. A.1), we introduce a generic time entity, `TimeUnit`. It ensures correct specification of time units throughout the whole data management process from creation of tariff data to its application in price calculation. Consider for example a price per week which should be specified as such in the tariff data. To arrive at monthly cost, it needs to be converted to months which usually means an assumption of four weeks per month. By providing explicit time periods, we can enter the time specification exactly as the provider defined it and implement explicit assumptions about conversion factors. This ensures consistent application of time periods for all uses of tariff data and allows more flexibility for calculation purposes.

Therefore, we implemented the following time periods in our system, which inherit from the parent type `TimeUnit`: `Year`, `Month`, `Week`, `Day`, `Hour`, `Minute`, `Second`. For a simple application, we assume 4 `Week`s or 30 `Day`s per `Month` to convert time periods. In a more advanced implementation, one might count the exact days for months when applied to specific calendar dates. By modeling the appropriate time period announced by the provider's tariff, we avoid loss of information (by converting all times to e.g. hours) and precise handling of time-related information.

B. OTM Application Listing

This section contains the full listing of the Java-style sample code of the OTM application on an AWS tariff. While the code gives a precise definition on the objects' states and relations, for practical applications a more high-level interface would support a more efficient conversion of the provider's different tariff attributes into the OTM.

```
// Define Provider and DcLocation
Provider aws = new Provider("Amazon Web Services);
DcLocation dcffm = new DcLocation("Frankfurt", "Germany");
aws.addDcLocation(dcffm);

// Define Resources
// VM
VmResource vm = new VmResource();
vm.setCpuCores(new CpuCore(2));
vm.setRamResource(new GB(7.5));

// Storage
StorageResource storage = new StorageResource();
storage.setSubtype(StorageResource.INTERNAL);
storage.setSize(new GB(32));
storage.setSsd(true);

storage.setValidValues(new ValueRange(new GB(32)));

// Data Transfer
TrafficResource inTraffic = new TrafficResource();
inTraffic.setDirection(TrafficResource.INGRESS);

TrafficResource outTraffic = new TrafficResource();
outTraffic.setDirection(TrafficResource.EGRESS);

// Operating System
OsResource os = new OsResource();
os.setFamily(OsResource.LINUX);
os.setVendor("AWS");
os.setVariant("Amazon AMI Linux");

// Static IP
NetworkResource ip = new NetworkResource();
ip.setSubType(NetworkResource.STATIC_IP);

// Define tariffs and prices
// Tariff definition
Tariff tariff = new Tariff("m3.large, reserved 1 year, Partial Upfront");
tariff.setProvider(aws);
dcffm.addTariff(tariff);

// VM TariffItem
TariffItem vmItem = new TariffItem();
vmItem.setResource(vm);

Price upfront = new Price("Upfront payment");
upfront.setCurrency("USD");
upfront.setNetAmount(492);
upfront.setBookingPeriod(new Year(1));

vmItem.addPrice(upfront);

Price hourly = new Price("Hourly rate");
```

```
55 hourly.setCurrency("USD");
56 hourly.setNetAmount(0.054)
57 hourly.setPerQuantity(new Count(1));
58 hourly.setPerTime(new Hour(1));
59 hourly.setBookingPeriod(new Year(1));
60
61 vmItem.addPrice(hourly);
62
63 tariff.addItem(vmItem);
64
65 // Storage TariffItem
66 TariffItem storageItem = new TariffItem();
67 storageItem.setResource(storage);
68
69 Price storagefree = new Price("Included storage");
70 storagefree.setCurrency("USD");
71 storagefree.setNetAmount(0);
72 storagefree.setBookingPeriod(new Year(1));
73 storageItem.addPrice(storagefree);
74
75 tariff.addItem(storageItem);
76
77 // Data Transfer TariffItem
78 TariffItem ingressItem = new TariffItem("Ingress");
79 ingressItem.setResource(inTraffic);
80 // Ingress is free
81 ingressItem.setName("Ingress, >0GB");
82 Price ingress = new Price("Free Incoming Traffic");
83 ingress.setCurrency("USD");
84 ingress.setNetAmount(0);
85 ingressItem.setPrice(ingress);
86 tariff.addItem(ingress);
87
88 // Egress
89 TariffItem egressItem = new TariffItem("Egress");
90 egressItem.setResource(outTraffic);
91
92 // Egress differs per quantity
93 Price firstgb = new Price("Egress, First GB");
94 firstgb.setCurency("USD");
95 firstgb.setNetAmount(0);
96 firstgb.setPerQuantity(new GB(1));
97 firstgb.setPerTime(new Month(1));
98 firstgb.setPartitionRange(new ValueRange<DataUnit>(new GB(0), new GB(1)))
       ;
99 egressItem.addPrice(firstgb);
100
101 Price upTo10Tb = new Price("Egress, 1GB-10TB");
102 upTo10Tb.setCurrency("USD");
103 upTo10Tb.setNetAmount(0.09);
104 upTo10Tb.setPerQuantity(new GB(1));
105 upTo10Tb.setPerMonth(new Month(1));
106 upTo10Tb.setPartitionRange(new ValueRange<DataUnit>(new GB(1), new TB(10)
       ));
```

```
egressItem.addPrice(upTo10Tb);

Price next40Tb = new Price("Egress, 10TB-50TB");
next40Tb.setCurrency("USD");
next40Tb.setNetAmount(0.085);
next40Tb.setPerQuantity(new GB(1));
next40Tb.setPerMonth(new Month(1));
next40Tb.setPartitionRange(new ValueRange<DataUnit>(new TB(10), new TB
    (50)));
egressItem.addPrice(next40Tb);

// ...and so on for all traffic price levels

// OS
TariffItem osItem = new TariffItem("OS");
osItem.setRessource(os);

Price osPrice = new Price("Standard Linux");
osPrice.setCurrency("USD");
osPrice.setNetAmount(0); // included

osItem.addPrice(osPrice);

// Static IP
TariffItem ipItem = new TariffItem("Static IP");
ipItem.setRessource(ip);

Price ipPrice = new Price("Static IP hourly surcharge");
ipPrice.setCurrency("USD");
ipPrice.setNetAmount(0.005);

ipItem.setPrice(ipPrice);

// Discount definition for VM revenue
Discount discount1 = new Discount("500k-4000k");
discount1.setSpendRange(new ValueRange<Double>(500000,4000000));
discount1.setDiscountFactor(0.05);
discount1.setScope(Discount.RESOURCE_LEVEL)

vmItem.addDiscount(discount1);

Discount discount2 = new Discount("4000k-10000k");
discount2.setSpendRange(new ValueRange<Double> (4000000,10000000));
discount2.setDiscountFactor(0.10);
discount2.setScope(Discount.RESOURCE_LEVEL);

vmItem.addDiscount(discount2);
```