Future Generation Computer Systems 29 (2013) 1000–1011

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Creating standardized products for electronic markets

Ivan Breskovic [a,*], Jörn Altmann [b], Ivona Brandic [a]

[a] Vienna University of Technology, Austria

[b] TEMEP, Seoul National University, South Korea

ARTICLE INFO

Article history:
Received 2 February 2012
Received in revised form
25 May 2012
Accepted 11 June 2012
Available online 18 June 2012

Keywords:
Service level agreement
SLA management
SLA matching
Electronic markets
Cloud economics
Autonomic computing
Standardized goods
Commodity goods
Market modeling
Market liquidity

ABSTRACT

Cloud computing is supposed to offer resources (i.e., data, software, and hardware services) in a manner similar to traditional utilities such as water, electricity, gas, and telephony. However, the current Cloud market is fragmented and static, preventing the successful implementation of ubiquitous computing on demand. In order to address the issue of fragmentation, commodity Cloud marketplaces have been suggested. However, as those marketplaces still suffer from being static (i.e., not being capable to adapt to changing market conditions and to the dynamics of user requirements for services), they do not operate at the optimal point. In this paper, we address this issue by channeling demand and supply into a few standardized services that are automatically adapted to user requirements in regular time intervals. For this, we utilize clustering algorithms and monitor the requirements of users. In order to avoid any cost to the user through these adaptations, we automatically adapt service level specifications of users to newly defined standardized goods. Using a simulation framework, we evaluate our approach and show its benefits to end users.

1. Introduction

Many large IT companies, such as eBay, Amazon, and Yahoo offer electronic marketplaces attracting millions of customers worldwide to buy and sell a broad variety of goods and services. This on-line trading model has many advantages over traditional marketplaces. Besides being fast, simple, and inexpensive, it allows users to place their bids anytime and from any geographical location that has Internet access.

Today, the Cloud computing paradigm (and utility computing in general) offers resources (i.e., software services, platform services, and hardware services) without regard to where the services are hosted or how they are delivered [1]. Following this idea, computing resources are supposed to be traded and delivered in a manner similar to utilities such as water, electricity, gas, and telephony [2]. However, the current Cloud market is fragmented and static, hindering the paradigm's ability to fulfill its promise of ubiquitous computing on demand.

Furthermore, very little research exists on the development of appropriate Cloud market platforms that are similar to marketplaces for electricity, water, and stocks [3–8]. Due to the large variability in services and still low number of traders, they often suffer from low liquidity (i.e., the ability to easily and quickly sell or purchase a service at a certain price), repelling potential consumers and disadvantaging new providers [9].

In order to create Cloud markets with sufficient stability and sustainability, the number of different types of Cloud computing resources needs to be low [9]. Some electronic Cloud marketplaces achieve this by offering a centralized place for search and by trading standardized Cloud services offered by various providers [7,10]. Standardization is achieved by creating a small number of services (products) derived from user requirements for services. However, even in this commodity resource Cloud market, the market is static, i.e., it cannot adapt seamlessly to changing market conditions, fluctuating trader base, or natural evolution of products.

In our approach, the method to address this issue is to channel the demand and supply into standardized Cloud products. We apply clustering algorithms to group similar user requirements for goods and apply adaptation methods for analyzing these requirements and for deciding whether to adapt the existing standardized products. By ensuring that standardized products are as close to user preferences as possible, the cost of searching for goods is reduced while the level of users' utility is at the highest level. This finally leads to an improved market liquidity, making the market more attractive to its users.

* Corresponding author.
E-mail addresses: breskovic@infosys.tuwien.ac.at (I. Breskovic), jorn.altmann@acm.org (J. Altmann), ivona@infosys.tuwien.ac.at (I. Brandic).

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.06.007

Besides, by considering the changes in user requirements (i.e., the constant changes in user base and actions users perform on the market), standardized products can be continuously updated. This way, the current market trend is always reflected without incurring any additional cost to users. Users trading old services are automatically switched to the new standardized product by the system. There is no need for users' manual adaptation of their services.

The main contributions of the paper can be summarized to: (1) the introduction of an approach for adaptation of standardized goods in Cloud markets; (2) an approach for automatic switching of user specifications for goods to newly created standardized products; (3) the formalization of a measure that is used for evaluating our approach and is based on determining the utility and the cost to users; and (4) a simulation of the proposed approach using an experimental testbed.

The remainder of the paper is organized as follows: Section 2 discusses related work. Section 3 provides a detailed explanation of our approach for automatic specification and adaptation of standardized goods in electronic Cloud markets. The implementation details are discussed in Section 4. Section 5 provides a description of the simulation testbed and presents the evaluation results. Finally, Section 6 concludes the paper.

2. Electronic markets in research

For positioning our work within the state-of-the-art, we briefly describe existing research on electronic markets. We classify the related work that is related to our work into three categories: (1) implementations of electronic markets for Grid and Cloud computing, (2) principles of autonomic computing to achieve sustainability in electronic markets, and (3) approaches for SLA matching.

2.1. Electronic markets for grid and cloud computing

Several research projects have discussed the implementation of system resource markets [3–5,7,8,11]. GRACE [3] developed a market architecture for Grid markets and outlined a market mechanism, while the good itself (i.e., computing resource) has not been defined. Moreover, the process of creating agreements between consumers and providers has not been addressed. The SORMA project also considered open Grid markets [4,5]. They identified several market requirements, such as allocative efficiency, budget-balance, truthfulness, and individual rationality [5]. However, they have not considered that a market can only function efficiently with a sufficiently large liquidity. In MACE [6], an abstract computing resource was defined that can be traded. However, a detailed specification of a good has not been given. GridEcon proposed a commodity market for Cloud computing services [7,8]. Although an explicit service level agreement for standardized Cloud services [10], the Cloud service requirements, and the requirements for trading have been defined and specified, the issue of adaptation of standardized goods has not been addressed. In the work on Cloud computing value chains [11], many important issues of electronic markets (e.g., improved Cloud pricing and licensing models) are discussed. However, while the diversity of virtualized resources was mentioned implicitly, the effect this diversity can have on the market has not been addressed.

2.2. Autonomic computing principles in electronic markets

As reported by Kephart and Chess [12], the scientific community has focused in recent years on making distributed systems adaptive and sustainable, referring to the principles of autonomic computing. However, most of the scientific work addresses technical issues to make systems autonomic, such as the development of negotiation protocols to make Cloud services self-adaptive [13], or considers using autonomic service management frameworks [14–16]. As a shortcoming of these works, they do not take economic methodologies into account. Research on autonomic systems focusing on economic methods and considerations is in its early stage [17]. For example, [18] propose mechanisms that are able to adaptively adjust their parameters based on the past behavior of participants. Another example is the self-organizing resource allocation mechanism for dynamic application layer networks [19]. However, they do not consider issues such as the adaptation of goods depending on the resources demanded or supplied in the market, which is a crucial element for autonomic marketplaces.

2.3. SLA matching

Specifications of user requirements in Clouds and Grids have been discussed by several research projects [20–22]. As reported in [23], most projects use SLA specifications based on Web service level agreement (WSLA) and WS-Agreement, which lack support for economic attributes and negotiation protocols, as well as some non-functional properties, such as security. To compensate for these shortcomings, extensions to WS-Agreement has been proposed [10]. Oldham and Verma introduce the use of semantic Web technologies based on Web service description language (WSDL-S) and Web ontology language (OWL) for enhancing WS-Agreement specifications to achieve autonomic SLA matching [20]. Similar to that, Green introduces an ontology-based SLA formalization where OWL and semantic web rule language (SWRL) are chosen to express the ontologies [22]. Dobson and Sanchez-Macian suggest producing a unified quality of service (QoS) ontology, applicable to the main scenarios such as QoS-based Web services selection, QoS monitoring, and QoS adaptation [21]. Another approach, which has been presented in [24], introduces an autonomic Grid architecture with mechanisms for dynamically reconfiguring service center infrastructures. It can be used to fulfill varying QoS requirements. Koller and Schubert discuss an autonomous QoS management, using a proxy-like approach for defining QoS parameters that a service has to maintain during its interaction with a specific customer [25]. Yarmolenko et al. make a case for increasing the expressiveness of SLAs [26]. This can possibly also increases market liquidity, if it comes to matching asks and bids and a common understanding of the SLA parameters has already been established. Our approach could be seen as complimentary in the sense that it makes sure that their precondition holds.

3. Adaptive standardized products for cloud markets

3.1. Specification of goods and requirements for goods

In our vision of Cloud markets, traders express their requirements and offers in form of *SLA templates*. A service level agreement (SLA) is a collection of service level requirements that have been negotiated and mutually agreed upon by a service provider and a service consumer. SLAs (as well as SLA templates) comprise SLA parameters where each parameter is defined through its *description* (i.e., its basic parameter properties (e.g., parameter name)), a *metric* that represents a method for calculating the parameter value, and the *service level objective* (SLO). The SLO specifies the contractual specified and agreed value of the parameter. Unlike SLA templates, SLAs are final documents signed between two trading parties. In order to create an SLA from an SLA template, the traders should

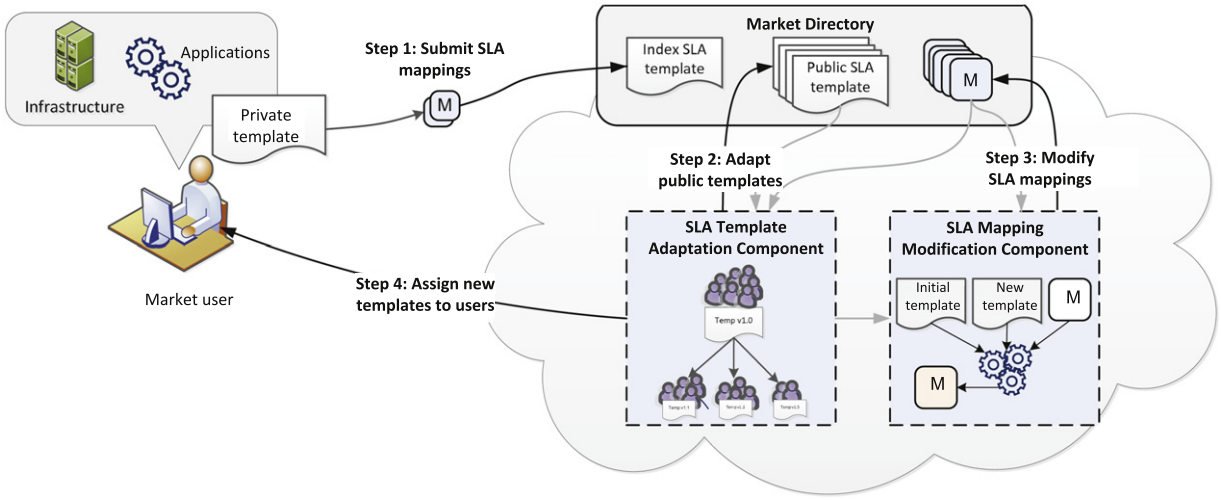


Fig. 1. The SLA mapping approach.

enter their legally required information, agree upon the final values of service objectives, and finally sign the contract.

Currently, SLAs are typically used for describing infrastructure services, i.e., in the infrastructure-as-a-service (IaaS) model, while for the other service models such as platform-as-a-service (PaaS) and software-as-a-service (SaaS) they are still not utilized. In this paper, we consider SLA parameters commonly used to describe infrastructure services [10].

We differentiate between two types of SLA templates in this context. On the one hand, user specifications of service requirements (i.e., consumer requests and provider offers) are expressed by means of *private SLA templates*. On the other hand, *public SLA templates*, which are stored in publicly available, searchable SLA registries, represent products that are traded on the market. Although these two SLA templates differ in their purposes, the only difference between their specifications are related to the SLO values. Namely, in private SLA templates, service level objectives are defined as ranges of acceptable values. For example, a private SLA template might state that any value between 80 and 100 for a parameter *CPUCores*, representing the number of processor cores, is acceptable for the user. This provides an additional level of flexibility, improving the probability of finding a match between a consumer's request and a provider's offer. Public SLA templates, on the other hand, specify exact values of SLOs (e.g., $CPUCores = 90$).

3.2. The SLA mapping approach

Fig. 1 depicts the process of trading on the Cloud market as assumed in this paper. In order to increase the market liquidity, the Cloud market only allows trading products that are described by public SLA templates (i.e., the standardized products). When entering the market to offer a service (in case of a service provider) or to request a service (in case of a service consumer), users must associate their services (as described by their private SLA templates) to the so-called index SLA template, which will be described later in this section. As SLA templates must be exactly the same in their structures for a successful trading between two parties, the SLA templates must be understood by both sides. This implies that users' private SLA templates must be equal to the public SLA templates and to the index SLA template, which is almost always not the case. To counteract this problem, we utilize the *SLA mapping technique* as a matching approach [27]. This is indicated as step 1 in Fig. 1.

SLA mappings are documents used to map the differences between two SLA templates by defining translations between differing properties of SLA parameters. Property translations can range from very simple, such as incompatible parameter names (e.g., different names used for the same SLA parameter) to complex translations, such as methods for calculating parameter values with different metrics (e.g., different units used to express the same parameter value).

SLA mappings are important, due to the lack of semantic descriptions of SLA parameters. Finding equivalent parameters in SLA templates is impossible unless a user has created a mapping to define this equality. Since standardized products (i.e., public SLA templates) are derived from users' service requirements (i.e., private SLA templates), understanding users' private SLA templates is crucial. Therefore, when entering the market, users are asked to create SLA mappings for their SLA templates (step 1 in Fig. 1).

For the purpose of creating SLA mappings, the service directory contains a specially formatted *index SLA template*, which is a public SLA template used as a registry rather than a trading product. The index template contains SLA parameters from all public SLA templates. When joining the market, users fetch the index template, check whether each of the SLA parameters of their private SLA templates exists, and if a parameter differs in the two templates, create an SLA mapping. In order to reduce the cost of this process, the service directory contains a collection of previously created SLA mappings, which can be reused by the users. If a user's private SLA template contains an SLA parameter that does not exist in the index template, the user can add it.

3.3. Automatic creation and adaptation of standardized products

Standardized Cloud products are created and later modified based on the requirements of the market participants (step 3 in Fig. 1). This process is executed automatically in certain time intervals and is achieved through the application of: (1) clustering algorithms to group similar user requirements (i.e., users' private SLA templates), and (2) adaptation methods to analyze the changes in user requirements.

The adaptation process is executed in three steps, as depicted in Fig. 2. First, clustering algorithms are applied to group structurally similar private SLA templates. Two SLA templates are similar by their structures, if parameter definitions and metrics of these templates are similar with respect to the distance function

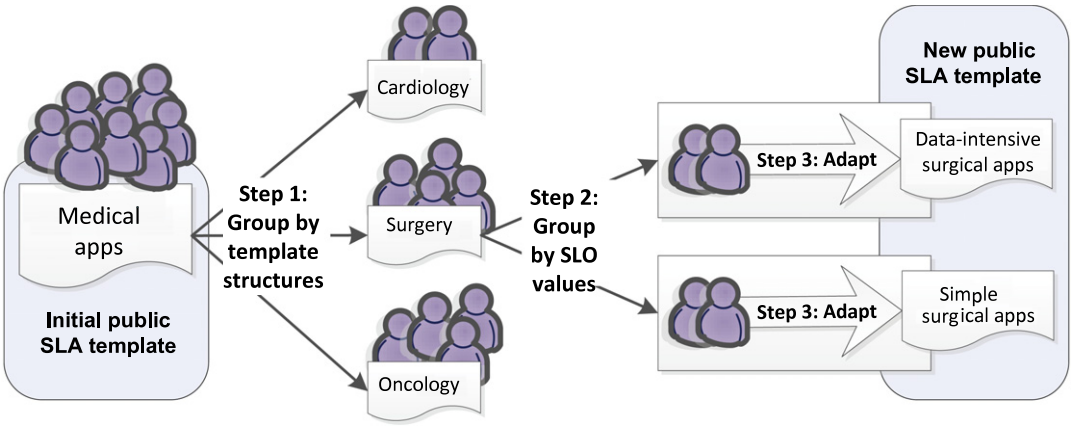


Fig. 2. Adaptation of SLA templates.

defined in Section 4.1.2. Often, these groups are subgroups of existing templates (i.e., branches of new products). For example, considering medical applications, one group of requirements might be for surgical applications, while an another one might be for services in oncology.

In the second step of the adaptation process, for each generated cluster, clustering algorithms are applied to create subgroups of requirements according to the SLO values expressed in users' private SLA templates. For example, in the group for surgical applications, a clustering algorithm might recognize a subgroup for data-intensive surgical applications (e.g., for big city hospitals), and another for simple surgical applications (e.g., for small ambulances).

For each of the subgroups of user requirements created by the second iteration of clustering, a new public SLA template is created. This is performed by applying an adaptation method (step 3 in Fig. 2)(step 2 in Fig. 1). The adaptation method determines for each SLA parameter from the current public SLA template whether its properties should be changed, a new parameter should be introduced, or an existing parameter should be deleted. After the new public SLA templates have been created, SLA mappings to the private SLA templates are created (step 3 in Fig. 1), the new public SLA templates published in the directory (step 3 in Fig. 1), the new public SLA templates are assigned to the user (step 4 in Fig. 1), and the old ones are deleted.

3.4. Adapting SLA mappings to the new public SLA templates

Although new public SLA templates reflect users' needs more precisely, users might prefer keeping the old public templates instead of using the new ones. This is because of the cost of creating new SLA mappings to the adapted public SLA templates, while the usage of the existing templates does not incur any additional cost. However, allowing users to utilize outdated templates results in a constant increase of the number of public SLA templates in the service directory. Consequentially, due to the increase in number of goods, this action increases the costs of searching for trading partner and implicitly has a negative impact on market liquidity. To prevent this, the market should enable users to utilize new public SLA templates without facing additional costs.

To achieve this goal, we investigate automatic modifications and creations of SLA mappings in this paper. Namely, we update SLA mappings for users by concatenate the existing SLA mapping with the same SLA mapping used to transform the current public SLA templates into the new public templates (step 4 in Fig. 1). As it will be shown in Section 5, this approach dramatically reduces the cost for users and enables the market to delete the old public SLA templates from the market directory, ensuring low cost of market maintenance.

4. Methods for automatic management of standardized products

In this section, we discuss the details of creating and adapting standardized products. First, we describe the clustering algorithms to group similar requirements of users (Section 4.1.1). Second, based on the requirements that define a cluster, we compute a new standardized product (i.e., a public SLA template) that will be the closest to the needs of the group of users. For this purpose, we utilize the adaptation methods (Section 4.2). Finally, after the new public SLA templates have been created, we apply an algorithm to automatically modify or create new SLA mappings to the new public templates so as to reduce the cost to users (Section 4.3).

4.1. Grouping private SLA templates

4.1.1. Application of clustering algorithms

For the purpose of grouping similar private SLA templates, we apply two clustering algorithms: DBSCAN and k-means. Note that user requirements, represented by their private SLA templates, are not stored in the market directory. Therefore, the algorithm for clustering user requirements must first reconstruct users' private SLA templates from the public SLA templates they utilize and the SLA mappings they submitted to the market.

DBSCAN is a data clustering algorithm that is based on the density distribution of nodes and finds an appropriate number of clusters [28]. The ϵ-neighborhood of a point p is defined as a set of points that are not farther away from p than a given distance ϵ. A point q is directly density-reachable from the point p if q is in the ϵ-neighborhood of p and the number of points in the ϵ-neighborhood of q is bigger than *MinPts*. A cluster satisfies two properties: (1) each two points are mutually density-reachable, and (2) if a point is mutually density-reachable to any point of the cluster, it is part of the cluster as well.

k-means is a clustering method that partitions N data points into k disjoint subsets S_j containing N_j data points so as to minimize the sum-of-square criterion $\sum_{j=1}^{k} \sum_{n \in S_j} d(x_n, \mu_j)^2$, where x_n is the nth data point, μ_j the geometric centroid of the data points in S_j, and d a distance function calculating the similarity of the two elements [29]. The algorithm, given an initial set of k means, assigns each data point to a cluster with the closest mean. It then calculates new means to be centroids of observations in the clusters and stops when the assignments no longer change the means. In our context, a data point is a user's private SLA template, and a cluster centroid is a new public SLA template for the group of users. A frequent problem in k-means algorithm is the estimation of the number k. Within this paper, we implement two approaches:

1. *Rule-of-Thumb* is a simple but very effective method for estimation the number k. The variable k is set to $\sqrt{N/2}$, where N is the number of entities [30].
2. *Hartigan's Index* is an internal index for scoring the number of clusters introduced in [31]. Let $W(k)$ represent the sum of squared distances between cluster members and their cluster centroid for k clusters. When grouping n items, the optimal number k is chosen so that the relative change of $W(k)$ multiplied with the correction index $\gamma(k) = n - k - 1$ does not significantly change for $k + 1$, i.e.,

$$H(k) = \gamma(k)\frac{W(k) - W(k+1)}{W(k+1)} < 10.$$

The threshold 10 shown in Hartigan's index is also used in our simulations. It is "a crude rule of thumb" suggested by Hartigan [31].

4.1.2. Computing distance between SLA templates

For determining the ϵ-neighborhood of a clustering point in case of DBSCAN and for computing the sum-of-square criterion in case of the k-means algorithm, we must define a function measuring distance (i.e., similarity) of two clustering elements (i.e., two SLA templates). Since clustering algorithms are applied for two different purposes, namely for grouping SLA templates based on their structures and based on their SLO values, we define two methods for computing the distance between two SLA templates. The first method is used with respect to the structure of SLA templates and the second one is used with respect to the SLO values of SLA templates.

The distance between the structures of SLA templates is expressed as a number of differences between parameter properties of two SLA templates. This value is calculated by iterating through each SLA parameter contained by at least one of the SLA templates and determining the distance for this SLA parameter. For two SLA templates T_1 and T_2, the distance between their structures S with respect to an SLA parameter p (i.e., its parameter description and metric) is defined as:

$$d_{S,p}(T_1, T_2) = \begin{cases} 0, & \text{if properties of } p \text{ are same in } T_1 \text{ and } T_2 \\ 1, & \text{if } T_1 \text{ or } T_2 \text{ does not contain } p \text{ or if only} \\ & \text{one property of } p \text{ differs in } T_1 \text{ and } T_2 \\ 2, & \text{if both properties of } p \text{ differ in } T_1 \text{ and } T_2. \end{cases}$$

Total distance between the two SLA templates with respect to their structures, noted $d_S(T_1, T_2)$, is calculated as the sum of distances between all SLA parameters contained in at least one of the templates.

When calculating the distance between two SLA templates with respect to their SLO values, we must consider that the SLO values in private SLA templates are represented by ranges of real numbers, as explained in Section 3.1. In order to compute the distance between such ranges, we utilize the Hausdorff metric [32], which defines this value as a maximum distance of one range to the nearest point in the other range. In detail, the Hausdorff distance $d_h(X, Y)$ between two non-empty subsets X and Y of a metric space M (in our case, two intervals in $\mathbb{R}$) is defined as:

$$d_h(X, Y) = \max\left\{\sup_{x\in X}\inf_{y\in Y} d(x, y), \sup_{y\in Y}\inf_{x\in X} d(x, y)\right\} \quad (1)$$

where sup represents the supremum, inf the infimum, and $d(x, y)$ any metric between points x and y. The Hausdorff distance is calculated by considering a point $x \in X$ and finding the least distance to a point $y \in Y$. This calculation is repeated for all $x \in X$ to find the maximum value. In the next step, the same process is performed with the roles of X and Y reversed. Finally, the largest of these two values is taken as the result.

Table 1
Sample SLA templates.

SLA template	SLA parameters
T_{init}	π : (*ResponseTime*, *second*, ⟨1, 3⟩) μ : (*ErrorRate*, *percentage*, ⟨0, 1⟩)
T_1	π : (*RespTime*, *second*, ⟨1, 4⟩) μ : (*ErrRate*, *percentage*, ⟨0, 2⟩)
T_2	π : (*ResponseTime*, *millisecond*, ⟨800, 3300⟩) μ : (*ErrorRate*, *percentage*, ⟨1, 3⟩)
T_3	π : (*RespTime*, *millisecond*, ⟨1100, 4500⟩) μ : (*Error*, *percentage*, ⟨0, 1⟩)

In our case, X and Y are values of service level objectives of an SLA parameter π from private SLA templates T_1 and T_2, noted $SLO(\pi)_{T_1}$ and $SLO(\pi)_{T_2}$. Since SLO values are unbounded sets of real numbers, $d(x, y)$ is the Euclidean distance between points x and y. Having said that, considering Eq. (1), for any two bounded ranges $SLO(\pi)_{T_1} = [x_1, x_2]$ and $SLO(\pi)_{T_2} = [y_1, y_2]$, it is simple to show that their Hausdorff distance is

$$d_h(SLO(\pi)_{T_1}, SLO(\pi)_{T_2}) = \max(|x_1 - y_1|, |x_2 - y_2|). \quad (2)$$

The total distance between two SLA templates with respect to their SLO values is calculated by computing distances between each SLO value defined in at least one of the templates and adding them up. Since SLO values of different SLA parameters are usually expressed in different measurement units, the values are not mutually comparable. Therefore, in order to compute the overall distance, SLO values are fitted into the range [0, 1] before summing. This is done by applying the following equation:

$$d_n(SLO(\pi)_{T_1}, SLO(\pi)_{T_2}) = \frac{d_h(SLO(\pi)_{T_1}, SLO(\pi)_{T_2}) - \min(d_h(SLO(\pi)))}{\max(d_h(SLO(\pi))) - \min(d_h(SLO(\pi)))} \quad (3)$$

where $d_h(SLO(\pi))$ is the set of all distances between all SLA templates with respect to the SLO value of the SLA parameter π. If a value is contained by only one of the two templates, the distance is maximum, i.e., equal to 1. In case the SLO values are expressed in a unit different from the SLO unit defined by the initial public SLA template, the value is being converted into the unit of the public template before computing the distance.

To demonstrate the methods for computing the distance between SLA templates, we use the following example. Table 1 depicts three private SLA templates T_1, T_2, and T_3 and an initial public SLA template T_{init} to which private templates are associated. Parameter definitions are given as tuples (*Name*, *Unit*, *SLO*), in which the elements represent the parameter name, the metric used, and the SLO value.

With respect to the SLA parameter π, private SLA templates T_1 and T_2 differ in both the parameter name and the parameter unit. Therefore, the distance between the template structures $d_{S,\pi}(T_1, T_2)$ is equal to 2. Since the specification of the parameter μ differs only in its name, the distance $d_{S,\mu}(T_1, T_2)$ is equal to 1. In total, the distance between structures of T_1 and T_2 is equal to $d_S(T_1, T_2) = d_{S,\pi}(T_1, T_2) + d_{S,\mu}(T_1, T_2) = 3$. Similarly, we get the distances $d_S(T_1, T_3) = 2$ and $d_S(T_2, T_3) = 2$.

Before computing the distances between the SLO values, they must be converted to the unit stated in the initial public SLA template. This is performed by applying the SLA mappings submitted by the users. Then, the distance between SLO values between T_1 and T_2 with respect to the SLA parameter π is calculated as:

$$d_h(SLO(\pi)_{T_1}, SLO(\pi)_{T_2}) = \max(|1 - 0.8|, |4 - 3.3|) = 0.7.$$

Similarly, the distances for the other SLA parameters and SLA templates can be calculated:

$$d_h(SLO(\pi)_{T_1}, SLO(\pi)_{T_3}) = 0.5$$
$$d_h(SLO(\pi)_{T_2}, SLO(\pi)_{T_3}) = 1.2$$
$$d_h(SLO(\mu)_{T_1}, SLO(\mu)_{T_2}) = 1$$
$$d_h(SLO(\mu)_{T_1}, SLO(\mu)_{T_3}) = 1$$
$$d_h(SLO(\mu)_{T_2}, SLO(\mu)_{T_3}) = 2.$$

After all Hausdorff distances have been calculated, they can be normalized using the Eq. (3):

$$d_n(SLO(\pi)_{T_1}, SLO(\pi)_{T_2}) = \frac{d_h(SLO(\pi)_{T_1}, SLO(\pi)_{T_2}) - \min(d_h(SLO(\pi)))}{\max(d_h(SLO(\pi))) - \min(d_h(SLO(\pi)))} = \frac{0.7 - 0.5}{1.2 - 0.5} = 0.285$$
$$d_n(SLO(\mu)_{T_1}, SLO(\mu)_{T_2}) = \frac{1 - 1}{2 - 1} = 0.$$

Finally, the total distance between SLA templates T_1 and T_2 with respect to the SLO values is

$$d_{SLO}(T_1, T_2) = d_n(SLO(\pi)_{T_1}, SLO(\pi)_{T_2}) + d_n(SLO(\mu)_{T_1}, SLO(\mu)_{T_2}) = 0.285.$$

4.2. Adaptation methods for the evolution of public SLA templates

For adapting the standardized products (i.e., the public SLA templates), so that they optimally reflect user requirements, we utilize three adaptation methods. For each public SLA template (in this process called *initial public SLA template*) and for each SLA parameter contained in the initial public SLA template, these methods determine whether the current parameter name and metric should be changed, a new parameter should be added, or an existing one deleted. This is performed by analyzing the distribution of parameter preferences of the users, who use the public SLA template. This analysis comprises sorting, classification, and counting of SLA mappings that users created for an SLA parameter. In particular, the adaptation methods apply selection criteria (which are specific to each of the methods), in order to find the SLA parameter value of each SLA parameter that are preferred by users. These values are then used to define a new public SLA template and replace the initial public SLA template.

To demonstrate the workings of these methods, we use the following example. We consider the evolution of an SLA parameter π occurring in an initial public SLA template T_{init}, when adapting the template based on SLA mappings of 100 users. The name and metric of the parameter π in the initial template is (*Price*, *EUR*). It is assumed that 20% of all users do not use the parameter π in their private SLA templates, and the rest of them define SLA mappings according to the distribution presented in Table 2. Note, the last column of the Table 2 represents the number of non-mapped values, i.e., the number of private SLA templates containing the same value as in the initial public SLA template. Table 3 represents the distribution of an SLA parameter μ, which exists in 75% of users' private SLA templates and does not exist in the initial public SLA template.

The **maximum method** selects the option that has the highest number of SLA mappings. This option is called the maximum candidate. The maximum candidate is then used in the new SLA template. If there is more than one maximum candidate with the same number of SLA mappings, one of them is chosen randomly. With respect to the given example, since the majority of users utilize the parameter π, it stays in the template. *Rate* becomes the new parameter name because of the highest number of mappings (40% in comparison to 30%, who want to keep the name *Price*) (i.e., who did not create mappings). The price will be expressed in Japanese Yens due to the highest number of mappings. The parameter μ will also be in the new template, since more than 50% of the users utilize it in their private SLA templates, and parameter properties will be (*MemoryConsumption*, *Gbit*).

Table 2
Distribution of mappings for the SLA parameter π.

(a) Name of the SLA parameter π				
Name used:	*Cost*	*Charge*	*Rate*	(*Price*)
Number of mappings: (%)	15	15	40	(30)
(b) Metric of the SLA parameter π				
Metric used:	*USD*	*GBP*	*YPI*	(*EUR*)
Number of mappings: (%)	38	2	40	(20)

Table 3
Distribution of mappings for the SLA parameter μ.

(a) Name of the SLA parameter μ			
Name used:	*MemoryConsumption*	*Consumption*	
Number of mappings: (%)	70	30	
(b) Metric of the SLA parameter μ			
Metric used:	*Mbit*	*Gbit*	*Tbit*
Number of mappings: (%)	5	55	40

In order to increase the requirements for selecting the maximum candidate, the **threshold method** introduces a threshold value. In this method, the property is chosen, if its property value is used more than the given threshold and has the highest count. If more than one parameter property value satisfies the two conditions, one of them is chosen randomly. Throughout the evaluation in this paper, we fix the threshold to be 60%. In the given example, neither the mappings for the name nor for the metric of the parameter π exceeds the threshold value. As for the new parameter μ, it will be represented in the updated public template according to the properties chosen by the maximum method.

The **significant-change method** changes an SLA parameter property value, only if the percentage difference between the maximum candidate and the current public SLA template value exceeds a given threshold, which we assume to be significant at $\sigma_T > 15\%$. In the given example, 40% of users have the name *Rate* for the parameter *Price*, while 30% of users use the same name as in the public SLA template. Since the percentage difference of 33% is higher than the given threshold, *Rate* will be chosen as the new name for the parameter. As the new metric, *YPI* will be chosen. As the parameter μ does not exist in the old public SLA template, the decision about this SLA parameter is made as described for the maximum method.

4.3. Automatic adaptation of SLA mappings to new public SLA templates

In order to reduce the cost of creating new SLA mappings for users and, therefore, make the market more attractive, users' SLA mappings are automatically redefined once a new public SLA template has been introduced. This way, users' existing SLA mappings can always be used for newly created public SLA templates.

For our discussion of the algorithm for autonomic SLA mapping, we consider the index public SLA template T_{index}, a user's private SLA template T_{user}, an initial public SLA template T_{init}, and a newly generated public SLA template T_{new}. An SLA parameter α of an SLA template T is defined by its description $D_T(\alpha)$, its metric $F_T(\alpha)$, and its SLO value $SLO_T(\alpha)$. It is denoted as $[D_T(\alpha), F_T(\alpha), SLO_T(\alpha)]$. An

1: **if** $\chi_{T_{new}}(\alpha) = \chi_{T_{init}}(\alpha) = false \vee$
$(\chi_{T_{new}}(\alpha) = \chi_{T_{init}}(\alpha) = true \wedge D_{T_{new}}(\alpha) = D_{T_{init}}(\alpha))$ **then**
2: keep identical SLA mappings
3: **else if** $\chi_{T_{new}}(\alpha) = \chi_{T_{init}}(\alpha) = \chi_{T_{user}}(\alpha) = true$ **then**
4: **if** $D_{T_{init}}(\alpha) \neq D_{T_{new}}(\alpha) \wedge D_{T_{init}}(\alpha) = D_{T_{user}}(\alpha)$ **then**
5: create $D_{T_{new}}(\alpha) \leftrightarrow D_{T_{user}}(\alpha)$
6: **else if** $D_{T_{new}}(\alpha) \neq D_{T_{init}}(\alpha) \neq D_{T_{user}}(\alpha)$ **then**
7: combine $D_{T_{new}}(\alpha) \leftrightarrow D_{T_{init}}(\alpha)$ and $D_{T_{init}}(\alpha) \leftrightarrow D_{T_{user}}(\alpha)$
8: **else if** $D_{T_{new}}(\alpha) \neq D_{T_{init}}(\alpha) \wedge D_{T_{new}}(\alpha) = D_{T_{user}}(\alpha)$ **then**
9: delete SLA mapping $D_{T_{init}}(\alpha) \leftrightarrow D_{T_{user}}(\alpha)$
10: **end if**
11: **else if** $\chi_{T_{new}}(\alpha) = false \wedge \chi_{T_{init}}(\alpha) = \chi_{T_{user}}(\alpha) = true$ **then**
12: warn user about limited usage of α
13: **if** $D_{T_{init}}(\alpha) \neq D_{T_{user}}(\alpha)$ **then**
14: delete existing SLA mapping $D_{T_{init}}(\alpha) \leftrightarrow D_{T_{user}}(\alpha)$
15: **end if**
16: **else if** $\chi_{T_{init}}(\alpha) = true \wedge \chi_{T_{new}}(\alpha) = \chi_{T_{user}}(\alpha) = false$ **then**
17: do nothing
18: **else if** $\chi_{T_{init}}(\alpha) = false \wedge \chi_{T_{new}}(\alpha) = true$ **then**
19: **if** $\chi_{T_{user}}(\alpha) = true$ **then**
20: combine $D_{T_{new}}(\alpha) \leftrightarrow D_{T_{index}}(\alpha)$ and $D_{index}(\alpha) \leftrightarrow D_{T_{user}}(\alpha)$
21: inform user about possible utilization of α
22: **else**
23: inform user about new parameter $[D_{T_{new}}(\alpha), F_{T_{new}}(\alpha), SLO_{T_{new}}(\alpha)]$
24: **end if**
25: **else if** $\chi_{T_{init}}(\alpha) = \chi_{T_{new}}(\alpha) = true \wedge \chi_{T_{user}}(\alpha) = false$ **then**
26: do nothing
27: **end if**

Fig. 3. Algorithm for autonomic SLA mapping modifications.

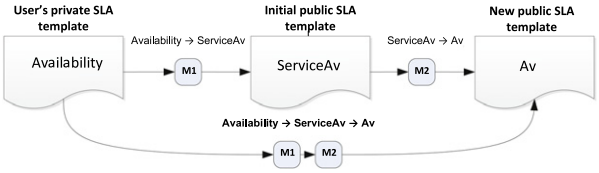


Fig. 4. Building an SLA mapping as a combination of two mappings.

SLA mapping between descriptions of an SLA parameter α in SLA templates T_1 and T_2 is denoted as $D_{T_1}(\alpha) \leftrightarrow D_{T_2}(\alpha)$. Additionally, we define a function χ to determine whether an SLA template T contains an SLA parameter α:

$$\chi_T(\alpha) = \begin{cases} \text{false}, & \text{if } T \text{ does not contain } \alpha \\ \text{true}, & \text{if } T \text{ contains } \alpha. \end{cases} \quad (4)$$

The algorithm iterates through all SLA mappings applied to transform T_{init} into T_{new}, and, for each mapping, it executes one of the possible transformation actions (Fig. 3). As a first step, the algorithm checks whether α exists in at least T_{init} or T_{new} and that its properties differ in those templates. In case that this does not hold, the existing user's SLA mappings for this parameter are kept identical (lines 1–2).

In case α exists in all observed SLA templates (line 3), one of the following actions is executed:

1. If a parameter property did not differ in T_{init} and T_{user}, but it changed in T_{new}, a new SLA mapping is created to map the newly created difference (lines 4–5).
2. If a parameter property differs in all three templates, a new SLA mapping is created (lines 6–7). It is a combination (concatenation) of two existing mappings so that the output of one becomes the input for the second mapping, as illustrated in Fig. 4.
3. If a parameter property does not differ in T_{user} and T_{new}, the existing SLA mapping is deleted (lines 8–9).

In case α was deleted from T_{init} but needed by the user, the algorithm informs the user about this action as the user will not be able to utilize the parameter anymore (lines 11–12). The algorithm also deletes a possibly existing SLA mapping for the deleted parameter (lines 13–14).

If the parameter was removed from the public SLA template, but the user does not need it, there is nothing to be executed (lines 16–17).

If a new parameter is introduced in T_{new} (line 18) and if the user's private SLA template contains that parameter (line 19), a new SLA mapping is created, which is a combination of two SLA mappings: (1) the SLA mapping between the property values stated in the user's private SLA template and the index template, and (2) the SLA mapping between the values stated in the new public SLA template and the index template (line 20). Besides creating a new SLA mapping, the algorithm also informs the user about the possibility of using an additional SLA parameter in the the market (line 21). Note, the SLA mapping to the parameter of the index SLA template exists, since the index template contains parameters from all public and private SLA templates and since all users create mappings to those parameters when entering the market. If a new SLA parameter is introduced in T_{new}, but the user does not have it in the private SLA template, the algorithm only informs the user about the possibility of using a new SLA parameter as soon as the user manually created an SLA mapping (lines 22–23).

Finally, if the parameter properties changed in T_{init} and T_{new}, but the user does not utilize the parameter, there is no need to perform any action as the changes in the parameter properties are not of the user's concern (lines 25–26).

Note, Fig. 3 deals with parameter descriptions only, while our implementation also considers their metrics. Our implementation simply achieves that by replacing D_T with F_T in the algorithm presented.

5. Evaluation

In this section, we present our simulation-based evaluation of the SLA mapping approach. In particular, we assess the benefits and the cost of creating adaptive standardized products. Moreover, we evaluate the algorithm for automated SLA mapping management and show its impact on users and the Cloud marketplace in general.

5.1. Simulation environment and testbed

For the simulation, we designed a framework for automated management of SLA mappings and generation of SLA templates. Fig. 5 depicts our simulation framework. The simulation framework comprises an electronic market (i.e., Cloud market) with a service directory that stores information about resources available on the market (i.e., public SLA templates), the index template, as well as the SLA mappings submitted by the users. The market is accessed using the frontend services for administration (e.g., creation of SLA templates), accounting (e.g., creation of user accounts), querying (e.g., search for an appropriate SLA template), and management of SLA mappings (e.g., definition of mappings for a user). Users (i.e., service consumers and providers) utilize the SLA mapping middleware to create and manage their SLA mappings to the index SLA template. The market self-adaptation cycle constantly performs monitoring of market performance (from both the economical and infrastructural perspectives) and modifies standardized products or derives new standardized products.

The simulation process is conducted in several steps, as depicted in Fig. 6. First, an initial public SLA template is created by the administrator. It could be based on a randomly created SLA template or a randomly picked private SLA template of a user. In the second step, a fixed number of service users fetch the index SLA template. Using this index SLA template, the users create SLA mappings to bridge the differences between their private SLA templates and the index SLA template (step 3). After all users have

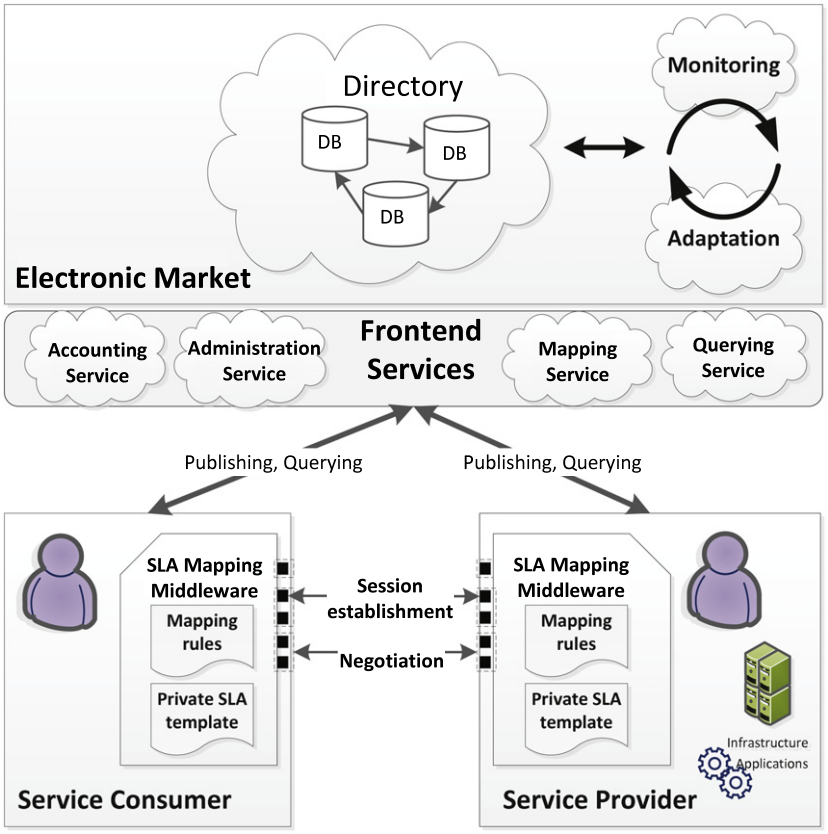


Fig. 5. Simulation environment.

submitted their SLA mappings, the adaptation process is started. Based on the result of the clustering algorithms and adaptation methods, the new public SLA templates are created (step 4). In the next steps, users' SLA mappings are modified (step 5) and new public SLA templates are assigned to the users (step 6), replacing the initial (i.e., currently existing) public SLA templates.

In our simulation, the initial public SLA template contains 8 SLA parameters. Users' private SLA templates are generated randomly at the beginning of the evaluation process and can contain up to 11 SLA parameters, where users can choose between 5 predefined parameter names and 4 metrics for each SLA parameter. Table 4 summarize the simulation settings.

For the sake of simplicity, the number of SLA parameters and variety of parameter definitions (i.e., parameter names and metrics) used in our simulations is relatively low compared to the SLAs commonly used to describe IaaS services in the real world applications. However, the motivation for our approach of standardizing Cloud services rises with the additional complexity of users' SLAs. Therefore, it is reasonable to expect that the approach presented in this paper would demonstrate even better results with the production SLAs. This will be examined in detail in our future work.

After new public SLA templates have been derived and users' SLA mappings adapted, the net utility is measured to evaluate not the entire SLA mapping process but rather the effects of the clustering and adaptation approach on the SLA mapping approach.

5.2. Formalization of the utility and cost model

For the evaluation of our clustering and adaptation approach, we define the cost and the benefits. Using the notation defined in Section 4.3, we define the utility function u^+ of a user *user* with respect to the difference of the properties of SLA parameter α of the public SLA templates and the users' private templates. The definition of the utility function follows the distance idea introduced in Section 4.1.2:

$$u^+_{user}(\alpha) = \begin{cases} 0, & \chi_{T_{user}}(\alpha) \neq \chi_{T_{new}}(\alpha) \vee \chi_{T_{user}}(\alpha) = \chi_{T_{new}}(\alpha) \\ & = \text{false} \\ A, & \chi_{T_{user}}(\alpha) = \chi_{T_{new}}(\alpha) = \text{true} \wedge \\ & D_{T_{user}}(\alpha) \neq D_{T_{new}}(\alpha) \wedge F_{T_{user}}(\alpha) \neq F_{T_{new}}(\alpha) \\ B, & \chi_{T_{user}}(\alpha) = \chi_{T_{new}}(\alpha) = \text{true} \wedge \\ & ((D_{T_{user}}(\alpha) = D_{T_{new}}(\alpha) \wedge F_{T_{user}}(\alpha) \neq F_{T_{new}}(\alpha)) \\ & \vee (D_{T_{user}}(\alpha) \neq N_{T_{new}}(\alpha) \wedge F_{T_{user}}(\alpha) = F_{T_{new}}(\alpha))) \\ C, & \chi_{T_{user}}(\alpha) = \chi_{T_{new}}(\alpha) = \text{true} \wedge \\ & D_{T_{user}}(\alpha) = D_{T_{new}}(\alpha) \wedge F_{T_{user}}(\alpha) = F_{T_{new}}(\alpha). \end{cases} \quad (5)$$

As stated in the definition of the utility function, the user gains utility for an SLA parameter, only if it can be utilized, i.e., if it exists in both their private SLA template and in the new public SLA template. The utility depends on the similarity of the specification of the SLA parameters in the two SLA templates: it is assumed to be *A*, if both the description and the metric of α differ in the user's private SLA template and the public SLA template; *B*, if only one of the parameter properties (i.e., either description or metric) differs; and *C*, if SLA templates do not differ with respect to the parameter.

Although the SLA mapping approach brings many benefits to the market participants, it also incurs cost. The cost is incurred when a public SLA template has been adapted and the user has to define and submit new SLA mappings for the changed parameter properties. The cost function u^- for a user *user* with respect to an

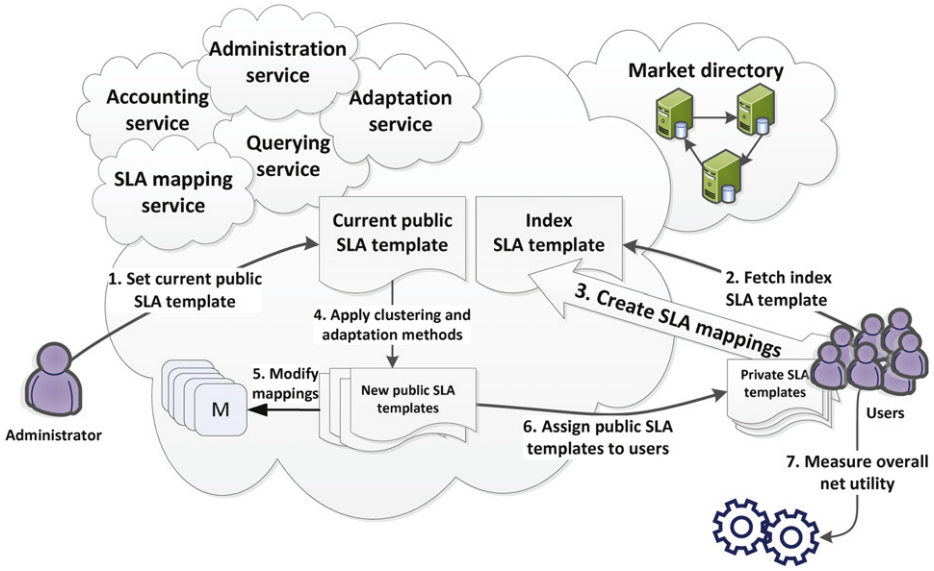


Fig. 6. Simulation testbed.

SLA parameter α is defined as follows:

$$u^{-}_{user}(\alpha) = \begin{cases} 0, & \chi_{T_{new}}(\alpha) = \chi_{T_{init}}(\alpha) = \text{false} \vee \chi_{T_{user}}(\alpha) = \text{false} \\ 0, & \chi_{T_{new}}(\alpha) = \chi_{T_{init}}(\alpha) = \chi_{T_{user}}(\alpha) = \text{true} \wedge (D_{T_{user}}(\alpha) = D_{T_{new}}(\alpha) \vee D_{T_{user}}(\alpha) = D_{T_{init}}(\alpha)) \wedge (F_{T_{user}}(\alpha) = F_{T_{new}}(\alpha) \vee F_{T_{user}}(\alpha) = F_{T_{init}}(\alpha)) \\ 0, & \chi_{T_{new}}(\alpha) = \chi_{T_{user}}(\alpha) = \text{true} \wedge \chi_{T_{init}}(\alpha) = \text{false} \wedge D_{T_{user}}(\alpha) = D_{T_{new}}(\alpha) \wedge F_{T_{user}}(\alpha) = F_{T_{new}}(\alpha) \\ D, & \chi_{T_{new}}(\alpha) = \chi_{T_{init}}(\alpha) = \chi_{T_{user}}(\alpha) = \text{true} \wedge ((D_{T_{user}}(\alpha) \neq D_{T_{new}}(\alpha) \wedge D_{T_{user}}(\alpha) \neq D_{T_{init}}(\alpha)) \vee (F_{T_{user}}(\alpha) \neq F_{T_{new}}(\alpha) \wedge F_{T_{user}}(\alpha) \neq F_{T_{init}}(\alpha))) \\ D, & \chi_{T_{new}}(\alpha) = \chi_{T_{user}}(\alpha) = \text{true} \wedge \chi_{T_{init}}(\alpha) = \text{false} \wedge (D_{T_{user}}(\alpha) \neq D_{T_{new}}(\alpha) \vee F_{T_{user}}(\alpha) \neq F_{T_{new}}(\alpha)). \end{cases} \quad (6)$$

The cost function specifies the cost incurred by the necessity of creating new SLA mappings. A user has no cost for an SLA parameter, if: (1) the parameter does not exist in the private SLA template or in neither the initial (current) SLA public template nor the new public SLA template; (2) the parameter properties of the user's private SLA template are the same as in the new public SLA template or as in the initial (current) public SLA template. In both cases, the SLA mappings have already been created; (3) the parameter was added to the new SLA template (i.e., the SLA parameter did not exist in the initial public SLA template), but its properties are the same as in the user's private SLA template. The user faces the cost D, if he must create SLA mappings for the parameter. This is the case if: (1) the parameter properties are different in the user's private SLA template and in the public SLA template; or (2) the parameter was added in the new public SLA template, but with the properties differing from those of the user's private SLA template.

Note, with the support of automated mapping, the cost of adaptation for the user would become quite low (Section 4.3). The user would only face cost when joining the system and when the mapping directory does not contain the mapping needed. However, independently of whether the cost is carried by the user or by the marketplace, the cost needs to be considered for objectively evaluating the clustering and adaptation approach.

Table 4
Simulation settings.

Parameter	Value
Number of service users (consumers and providers)	$100 \leq n \leq 10000$
Number of initial (currently existing) SLA templates	1
Number of parameters in initial SLA template	8
Number of parameters in private SLA templates	≤ 11
Number of different parameters considered at most	11
Size of the set of possible parameter names per SLA parameter	5
Size of the set of possible metrics per SLA parameter	4

We define the overall utility U^{+} and the overall cost U^{-} for all users C as:

$$U^{+} = \sum_{user \in C} \sum_{\alpha \in P} u^{+}_{user}(\alpha), \qquad U^{-} = \sum_{user \in C} \sum_{\alpha \in P} u^{-}_{user}(\alpha)$$

where P is the set of all SLA parameters occurring in T_{user}, T_{init} or T_{new}. The overall net utility U^{O} is then calculated as

$$U^{O} = U^{+} - U^{-}. \quad (7)$$

The return values of the utility function and the cost function are not strictly defined. However, considering the type of efforts and benefits, it should hold that $0 \leq A \leq B \leq C$ and $0 \leq D$. For our simulations, we fix these return values at $A = 1, B = 2, C = 3$ and $D = 1$.

5.3. Evaluation results

5.3.1. Creation and management of adaptive virtual products

For the cost-benefit evaluation of our approach, we use the overall net utility as defined with Eq. (7). For the first evaluation, we compare the net utility of our proposed approach (for applying clustering algorithms to group user requirements for the creation of new standardized products) with the net utility of the basic SLA mapping as described in [27]. In [27], only the adaptation methods were applied to create a single new public SLA template for all market participants. The comparison is shown in Fig. 7.

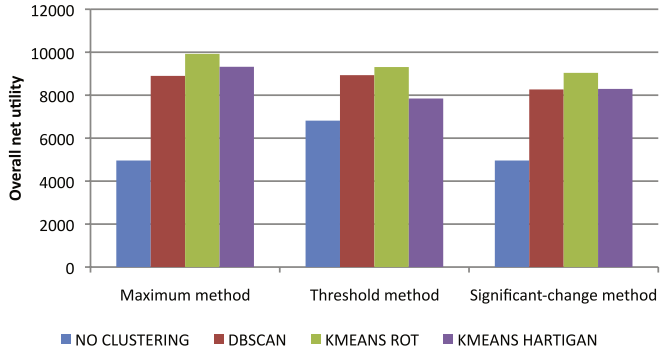


Fig. 7. Overall net utility (without automatically modifying users' SLA mappings).

Table 5
Number of generated clusters per algorithm and per number of users.

Users	DBSCAN	*k*-means (RoT)	*k*-means (H)
100	5	7	3
200	5	10	3
300	6	12	3
500	6	15	5
1,000	7	22	6
2,000	7	31	6
5,000	8	50	6
10,000	10	69	10

The first bar of each of the three sets of bars of Fig. 7 represents the overall net utility achieved for the case that only a single new public template using the basic SLA mapping has been used. The other three bars represent the overall net utility achieved by utilizing the three clustering algorithms (i.e., DBSCAN, *k*-means with rule-of-thumb, and *k*-means with Hartigan's index). As the comparison shows, the overall net utility, which is obtained by generating only one public SLA template for all market users with the basic mapping, is significantly lower than the overall net utility obtained with the clustering algorithms. This is due to the fact that many new public SLA templates are created that differ less from the users' private templates, reflecting users' needs more precisely.

When comparing the overall net utility gained by utilizing different adaptation methods, we can conclude that the best results are achieved by using the maximum method. Although the threshold method causes significantly lower number of changes (due to the high threshold) and, therefore, very low cost, the maximum method adapts the public templates more frequently and, therefore, achieves a high utility. The highest overall net utility is achieved by the *k*-means clustering algorithm with the rule-of-thumb method, due to the creation of a large number of very specific clusters.

Concluding, the utility rate highly depends on the number of generated clusters. The more clusters are created, the higher the utility will be. This relation can also be seen when comparing the net utility shown in Fig. 7 with the actual number of clusters. Table 5 shows the number of clusters per clustering algorithm, depending on the number of users creating SLA mappings. The maximum rate of overall net utility for n users can be achieved by creating n clusters, i.e., by generating one public SLA template per user. In this case, the public SLA templates would be equal to the users' private templates. The utility rate would in this case be maximum and the cost would be equal to 0. However, by raising the number of products on the market, i.e., the number of new public SLA templates, market liquidity is at the lowest point. Therefore, it is necessary to find a balance between keeping low cost of creating new SLA mappings and ensuring high liquidity of market goods. The problem of finding the optimal number of new public SLA templates to be introduced to the market is not in the scope of this paper and will be addressed in our future work.

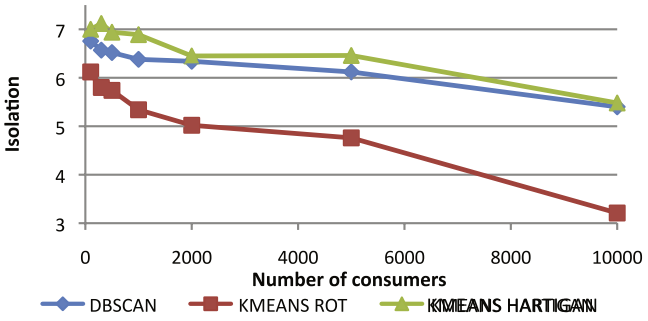


Fig. 8. Isolation of the new public SLA templates.

In order to show that the clusters generated are significantly different, we use the distance measure of Section 4.1.2 for calculating the isolation between clusters. Isolation specifies the average difference between newly generated public SLA templates. The larger the value of the isolation measure, the more distinct the SLA templates are. We imply that a high isolation of public SLA templates ensures more distinct products and, therefore, provides better chances for creating product niches.

To introduce the measures of isolation, we define the following variables: C represents a set of clusters, where $C_j \in C$ is a cluster and C_{jc} is its centroid (i.e., the public SLA template). R_i is a clustering item and $D(R_i, R_k)$ is the distance between two clustering items. $|C|$ represents the total number of clusters and $|C_i|$ represents a number of items in a cluster C_i.

$$\frac{1}{|C|(|C|-1)} \sum_{C_i \in C} \sum_{C_k \in C} D(C_{ic}, C_{kc}). \tag{8}$$

Using this definition and the distance measure of Section 4.1.2, the following isolation rates are obtained for the clusters (Fig. 8). We have simulated SLA mappings specified by a varying number of users, ranging from 100 to 10,000 users. The adaptation method used for the evaluation is the maximum method.

A high isolation can be achieved by creating a small number of very distinct clusters. However, by doing so, the utility is reduced (Fig. 7). The clusters contain a larger number of items with a broader variety of preferences. Therefore, it is necessary to find a balance between creating more general clusters with high isolation (i.e., distinct market products) and well-formed clusters with respect to their utility.

As it can be seen in Fig. 8, the *k*-means algorithm with the Hartigan's index and the DBSCAN algorithm achieve a high rate of isolation. This result is not surprising, since both algorithms create a small number of clusters and, as presented in Fig. 7, achieve a lower overall net utility than the *k*-means algorithm with rule-of-thumb. Finally, the *k*-means with the rule-of-thumb creates clusters with low isolation, due to creating a large number of clusters.

To conclude, considering the overall net utility and the isolation, the best results are achieved by the *k*-means clustering algorithm with the rule-of-thumb method for determining the number of clusters. As depicted in Fig. 7, this algorithm achieves the highest net utility for all adaptation methods. This means that the newly created standardized products are created with the highest similarity to the user requirements. However, this comes with the cost of creating a large number of public SLA templates, which results in lower market liquidity. When compared with the *k*-means clustering algorithm with the Hartigan's index, it is more cost efficient, since it's computing complexity is significantly lower. In particular, unlike Hartigan's index, which requires several

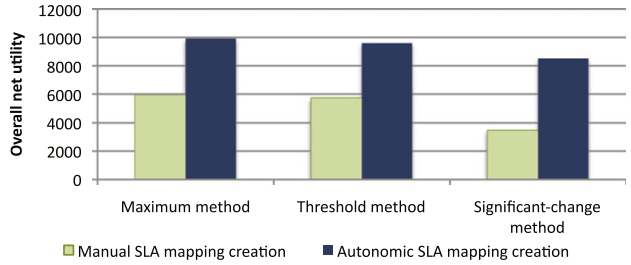


Fig. 9. Comparison of the overall net utility of the *k*-means Rule-of-Thumb clustering algorithm with and without automatic modification of users' SLA mappings.

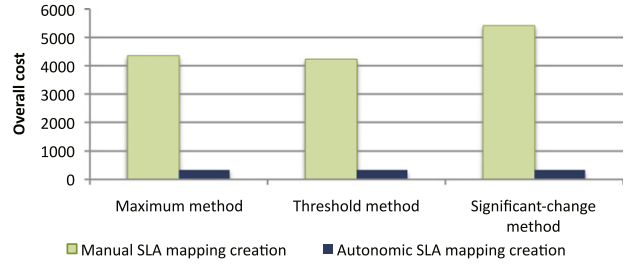


Fig. 10. Comparison of the overall cost of the *k*-means Rule-of-Thumb clustering algorithm with and without automatic modification of users' SLA mappings.

iterations of the *k*-means algorithm before finding the optimal number *k*, the rule-of-thumb method determines the number of clusters a priori. When compared to DBSCAN, it achieves significantly higher rate of the overall net utility. Having said that, we conclude that the *k*-means algorithm with the rule-of-thumb is the most appropriate method for creating standardized virtual products.

5.3.2. Automatic management of SLA mappings

By not considering the cost of creating and adapting SLA mappings for the users, the cost of the basic SLA mapping approach is strongly reduced. In this section, in particular, we assess the benefits of the automation and compare it with the results presented in the previous section.

Fig. 9 illustrates the overall net utility achieved by each of the adaptation methods with and without applying the algorithm for the automatic management of SLA mappings. For this purpose, we utilize only the *k*-means clustering algorithm with the rule-of-thumb, due to the best performance when compared to other clustering algorithms (Section 5.3.1). As also depicted, by applying the automation algorithm, the overall net utility is significantly higher than the overall net utility for the case when users must manually create new SLA mappings. Note, the overall utility does not differ for the two approaches, as the newly created public SLA templates are equal. The difference in the overall net utility comes from the reduction of the cost for creating SLA mappings.

The difference between the cost is depicted in Fig. 10. Since the users are not required to create any new SLA mappings when applying the algorithm for automatic SLA mapping management, the overall cost for users is reduced to 0.

It is important to note that the cost for creating SLA mappings has not vanished but carried by the marketplace instead of the users. However, through this approach, the human interaction is significantly reduced and, therefore, the cost for creating SLA mappings becomes negligible. Moreover, the computing complexity of the algorithm for automated management of SLA mappings is very low.

Concluding, the simulation results show the clustering approach for defining several public SLA templates is superior to a single adaptive public SLA template. In combination with the adaptation methods, it could be shown that the maximum method together with *k*-means algorithm with rule-of-thumb is the best combination. The overall net utility is the highest compared to other combinations. The cost can be reduced even further through the use of automated management of SLA mappings.

6. Conclusion

A large body of research into the Cloud paradigm yielded the technological development of Cloud infrastructures, such as development of the appropriate resource management models [33,34,8], solutions for the energy efficient management of Clouds [35], as well as security and privacy solutions [36]. Yet, very little research exists on the development of appropriate marketplaces in a similar way to commodities like energy, water, and gold [7]. In this paper, we have demonstrated means to make the computing resource market adaptable to changes in market demand and supply. Such a flexible commodity market is necessary in the high-tech industry environment, as the technological development of computing resources has a significant impact on the goods being available in the market.

In particular, in this paper, we have presented our idea of standardizing virtual services in electronic markets. We applied clustering algorithms to group similar user requirements for services and to create a limited number of adaptive standardized products with the goal of increasing the market liquidity.

In order to avoid cost for adapting existing SLA matchings with the SLA mapping approach, we have presented a cost-efficient method for automatically modifying users' SLA mappings needed for trading. Using the automated management of SLA mappings, public SLA templates can be deleted and replaced by the newly created public SLA templates without any problems. It reduces cost for maintenance and storage of public SLA templates. This is not possible with the manual approach, as it is necessity to create new SLA mappings before utilizing newly created public templates. This, in turn, is hard to achieve, as users prefer to keep their old public SLA templates to avoid additional cost of creating new SLA mappings.

Our evaluation based on a simulation framework showed that our clustering approach with different adaptation methods increases the overall net utility of traders. The *k*-means clustering algorithm with the rule-of-thumb has been identified as the best choice.

In our future work, we will explore methods for measuring market liquidity and will adapt the number of public SLA templates such that it increases the liquidity of the market to its maximum point.

Acknowledgments

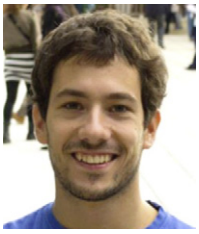
The authors would like to thank Michael Maurer and Vincent C. Emeakaroha for their contribution. The work described in this paper has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT08-018 and by the National Research Foundation of the Ministry of Education, Science and Technology of Korea under the grant K21001001625-10B1300-03310.

References

[1] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, M. Zaharia, Above the clouds: a Berkeley view of cloud computing, Tech. rep., EECS Department, University of California, Berkeley, February 2009.

[2] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility, Future Generation Computer Systems 25 (2009) 599–616.

[3] R. Buyya, D. Abramson, J. Giddy, A case for economy grid architecture for service oriented grid computing, Parallel and Distributed Processing Symposium 2.
[4] J. Nimis, A. Anandasivam, N. Borissov, G. Smith, D. Neumann, N. Wirstrm, E. Rosenberg, M. Villa, SORMA: business cases for an open grid market: concept and implementation, in: Grid Economics and Business Models, in: Lecture Notes in Computer Science, vol. 5206, Springer, Berlin, Heidelberg, 2008, pp. 173–184.
[5] D. Neumann, J. Stösser, C. Weinhardt, Bridging the adoption gap-developing a roadmap for trading in grids, Electronic Markets 18 (2008) 65–74.
[6] B. Schnizler, D. Neumann, D. Veit, C. Weinhardt, Trading grid services — a multi-attribute combinatorial approach, European Journal of Operational Research 187 (3) (2008) 943–961.
[7] J. Altmann, C. Courcoubetis, M. Risch, A marketplace and its market mechanism for trading commoditized computing resources, Annals of Telecommunications 65 (2010) 653–667.
[8] M. Risch, J. Altmann, L. Guo, A. Fleming, C. Courcoubetis, The gridecon platform: a business scenario testbed for commercial cloud services, in: Proceedings of the 6th International Workshop on Grid Economics and Business Models, GECON '09, Springer-Verlag, 2009, pp. 46–59.
[9] M. Risch, I. Brandic, J. Altmann, Using SLA mapping to increase market liquidity, in: Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops, in: Lecture Notes in Computer Science, vol. 6275, Springer, 2010, pp. 238–247.
[10] M. Risch, J. Altmann, Enabling open cloud markets through WS-agreement extensions, in: P. Wieder, R. Yahyapour, W. Ziegler (Eds.), Grids and Service-Oriented Architectures for Service Level Agreements, Springer, US, 2010, pp. 105–117.
[11] A.B. Mohammed, J. Altmann, J. Hwang, Cloud computing value chains: Understanding businesses and value creation in the cloud, in: D. Neumann, M. Baker, J. Altmann, O. Rana (Eds.), Economic Models and Algorithms for Distributed Systems, Autonomic Systems, Birkhäuser, Basel, 2010, pp. 187–208.
[12] J.O. Kephart, D.M. Chess, The vision of autonomic computing, Computer 36 (2003) 41–50.
[13] I. Brandic, D. Music, S. Dustdar, Service mediation and negotiation bootstrapping as first achievements towards self-adaptable grid and cloud services, in: Grids meet Autonomic Computing Workshop 2009. In conjunction with the 6th International Conference on Autonomic Computing and Communications, ACM, Barcelona, Spain, 2009, pp. 1–8.
[14] Y. Cheng, A. Leon-Garcia, I. Foster, Toward an autonomic service management framework: a holistic vision of SOA, AON, and autonomic computing, Communications Magazine, IEEE 46 (5) (2008) 138–146. http://dx.doi.org/10.1109/MCOM.2008.4511662.
[15] W.H. Oyenan, S.A. Deloach, Towards a systematic approach for designing autonomic systems, Web Intelligence and Agent Systems 8 (2010) 79–97.
[16] C. Lee, J. Suzuki, An autonomic adaptation mechanism for decentralized grid applications, in: Consumer Communications and Networking Conference, 2006. CCNC 2006. 3rd IEEE, vol. 1, 2006, pp. 583–589. http://dx.doi.org/10.1109/CCNC.2006.1593091.
[17] G. Cheliotis, C. Kenyon, Autonomic economics, in: E-Commerce, 2003. CEC 2003. IEEE International Conference on, 2003, pp. 120–127. http://dx.doi.org/10.1109/COEC.2003.1210241.
[18] D. Pardoe, P. Stone, M. Saar-Tsechansky, K. Tomak, Adaptive mechanism design: a metalearning approach, in: Proceedings of the 8th International Conference on Electronic Commerce, ICEC '06, ACM, 2006, pp. 92–102.
[19] W. Streitberger, T. Eymann, A simulation of an economic, self-organising resource allocation approach for application layer networks, Computer Networks 53 (2009) 1760–1770.
[20] N. Oldham, K. Verma, Semantic WS-agreement partner selection, in: 15th International Conference on World Wide Web. WWW '06, ACM Press, 2006, pp. 697–706.
[21] G. Dobson, A. Sanchez-Macian, Towards unified QoS/SLA ontologies, in: Services Computing Workshops, 2006. SCW '06, IEEE, 2006, pp. 169–174.
[22] L. Green, Service level agreements: an ontological approach, in: 8th International Conference on Electronic Commerce, ICEC '06, ACM, 2006, pp. 185–194.
[23] P. Karänke, S. Kirn, Service level agreements: an evaluation from a business application perspective, in: eChallenges e-2007, IEEE-CS Press, 2007, pp. 104–111.
[24] D. Ardagna, G. Giunta, N. Ingraffia, R. Mir, B. Pernici, Qos-driven web services selection in autonomic grid environments, in: In OTM Conferences, 2, 2006, pp. 1273–1289.
[25] B. Koller, L. Schubert, Towards autonomous SLA management using a proxy-like approach, Multiagent Grid Syst. 3 (2007) 313–325.
[26] V. Yarmolenko, R. Sakellariou, Towards increased expressiveness in service level agreements: Research articles, Concurrency and Computation: Practice and Experience 19 (2007) 1975–1990. http://dx.doi.org/10.1002/cpe.v19:14.
[27] M. Maurer, V.C. Emeakaroha, I. Brandic, J. Altmann, Cost-benefit analysis of an SLA mapping approach for defining standardized cloud computing goods, Future Generation Computing Systems 28 (1) (2012) 39–47.
[28] M. Ester, H.P. Kriegel, J. Sander, X. Xu, A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise, AAAI Press, 1996, pp. 226–231.
[29] J.B. MacQueen, Some methods for classification and analysis of multivariate observations, in: 5th Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, University of California Press, 1967, pp. 281–297.
[30] K.V. Mardia, J.T. Kent, Multivariate Analysis, Academic Press, 1980.
[31] J.A. Hartigan, Clustering Algorithms, John Wiley & Sons Inc, 1975.
[32] G. Rote, Computing the minimum Hausdorff distance between two point sets on a line under translation, Information Processing Letters 38 (1991) 123–127.
[33] R. Buyya, A. Sulistio, Service and utility oriented computing systems: Challenges and opportunities for modeling and simulation communities, in: Annual Simulation Symposium, 2008, pp. 68 –81. http://dx.doi.org/10.1109/ANSS-41.2008.35.
[34] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F.D. Rose, R. Buyya, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software — Practice and Experience http://dx.doi.org/10.1002/spe.995.
[35] O. A, L. Lefevre, When clouds become green: the green open cloud architecture, in: ParCo 2009, International Conference on Parallel Computing, 2009, pp. 228–237.
[36] I. Brandic, T. Anstett, D. Schumm, F. Leymann, S. Dustdar, R. Konrad, Compliant cloud computing (c3): Architecture and language support for user-driven compliance management in clouds, in: IEEE Cloud, 2010.



Ivan Breskovic is a Ph.D. student and research assistant at the Distributed Systems Group, Institute of Information Systems, Vienna University of Technology. He did his bachelor study in Software Engineering at the University of Zagreb, Faculty of Electrical Engineering and Computing in Zagreb, Croatia and master's study in Software Engineering and Information Systems at the same faculty. He is currently involved in the Austrian national FoSII (Foundations of Self-governing ICT Infrastructures) project funded by the Vienna Science and Technology Fund (WWTF). His areas of interest include cloud computing, cloud economics, autonomic computing, service level agreements and quality of service management.

Jörn Altmann is Associate Professor for Technology Management, Economics, and Policy at the College of Engineering of Seoul National University. Prior to this, he taught computer networks at the University of California at Berkeley, worked as a Senior Scientist at Hewlett-Packard Labs, and has been a postdoc at EECS and ICSI of UC Berkeley. During that time he worked on international research projects about pricing of network services. Dr. Altmann received his B.Sc. degree, his M.Sc. degree (1993), and his Ph.D. (1996) from the University of Erlangen-Nürnberg, Germany. Dr. Altmann's current research centers on the economics of Internet services and Internet infrastructures, integrating economic models into distributed systems. On these topics of research, he has major publications in conferences and journals, serves on editorial bodies of journals, is involved in many conference program committees on cloud computing, and has been an invited speaker to several workshops. He also served on several European, US American (National Science Foundation), and different national panels for evaluating research proposals on next generation networks and emerging technologies.

Ivona Brandic is Assistant Professor at the Distributed Systems Group, Information Systems Institute, Vienna University of Technology. Prior to that, she was Assistant Professor at the Department of Scientific Computing, Vienna University. She received her Ph.D. degree from Vienna University of Technology in 2007. From 2003 to 2007 she participated in the special research project AURORA (Advanced Models, Applications and Software Systems for High Performance Computing) and the European Union's GEMSS (Grid-Enabled Medical Simulation Services) project. She is involved in the European Union's SCube project and she is leading the Austrian national FoSII (Foundations of Self-governing ICT Infrastructures) project funded by the Vienna Science and Technology Fund (WWTF). She is a Management Committee member of the European Commission's COST Action on Energy Efficient Large Scale Distributed Systems. From June–August 2008 she was visiting researcher at the University of Melbourne. Her interests comprise SLA and QoS management, Service-oriented architectures, autonomic computing, workflow management, and large scale distributed systems (Cloud, Grid, Cluster, etc.).