2011 Fourth IEEE International Conference on Utility and Cloud Computing

Cost-Efficient Utilization of Public SLA Templates in Autonomic Cloud Markets

Ivan Breskovic, Michael Maurer, Vincent C. Emeakaroha, Ivona Brandic, Schahram Dustdar
Distributed Systems Group, Institute of Information Systems, Vienna University of Technology, Austria
{*breskovic, maurer, vincent, ivona, dustdar*}*@infosys.tuwien.ac.at*

***Abstract*—Dynamic and self-adaptable markets are fundamental for the successful implementation of the Cloud computing paradigm where computing resources are provided on demand and dynamically to the heterogeneous user base. Usually, in Cloud markets, contracts between traders are established using Service Level Agreements (SLAs), which include objectives of service usage. However, Cloud markets face challenges, which are usually not evident in other market types, such as non-standardized goods and large variety of resource types offered on the market, resulting in low number of matches of consumers' bids and providers' asks. Moreover, existing Cloud market mechanisms are usually static and cannot react on dynamic change of user requirements. To counteract these problems, we propose an adaptive approach for autonomically deriving public SLA templates (i.e., templates for generation of electronic contracts) based on user requirements. Although this approach brings many benefits, it also incurs cost to users, since they must adapt specifications of their requirements in order to match automatically generated public SLA templates. In this paper, we reduce this cost by autonomically modifying specifications of user requirements. We introduce a method for adapting public SLA templates not only in respect to definitions of SLA attributes, but also to attribute values. Based on an appropriate utility and cost model, we evaluate the approach and show that it significantly improves the performance of the adaptive SLA template technique.**

***Keywords*-Service Level Agreement; SLA management; SLA mapping; Cloud economics; electronic markets**

I. Introduction

Cloud computing is a novel computing paradigm where resources (e.g., software services and platforms) are accessed based on user requirements without regard to where the services are hosted or how they are delivered. Information is, therefore, commoditized and delivered in a manner similar to traditional utilities such as water, electricity, gas, and telephony [1]. Requirements of Cloud services are expressed and negotiated by means of Service Level Agreements (SLAs), which are contracts signed between a customer and a service provider.

For the successful implementation of the Cloud computing paradigm, well-developed software and hardware resource markets are fundamental [2]. They allow millions of customers worldwide to buy and sell consumable services at any time of the day and from any geographical location that has Internet access. Due to the broad scope of services and a large number of users trading, they often offer a potential for a relatively low price. Finally, they represent a simple, fast, and inexpensive way to sell and purchase Cloud services.

Many electronic marketplaces suffer from challenging situations [3]. Namely, due to the static nature of existing Cloud markets, they cannot react on dynamic change of user requirements. Moreover, due to the large variability in resource types and still low number of traders, markets suffer from low liquidity (i.e., probability to easily and quickly sell or purchase a service at a certain price), repelling potential consumers and providers. To counteract this problem, the SLA mapping approach was introduced [4], making a first step towards autonomic Cloud markets that adapt to changes in user demands. The approach utilizes *public SLA templates* to represent groups of products traded on the market, and *private SLA templates* to describe user requirements. Users (providers and consumers) associate their services to public SLA templates and submit *SLA mappings* to bridge the differences between their private SLA templates and the associated public SLA templates. Using the submitted mappings, user requirements are analyzed and new public SLA templates derived. This is done by applying clustering algorithms to group similar demands, and learning methods to adapt existing templates to reflect the needs of their users.

As shown in [4], the SLA mapping approach facilitates continuous market evolution and adaptation of public SLA templates, responding to market trends and changes. However, the approach has also several shortcomings. Namely, after the new public SLA templates are derived, users must create additional SLA mappings before using them, which incurs cost [3]. As a result, although new public SLA templates reflect their requirements in a larger extent, users might prefer keeping the old public SLA templates in order to avoid additional cost. This results in a constantly increasing number of public SLA templates on the market, increasing the cost of market maintenance and reducing the market liquidity, due to channeling demand through a larger number of public SLA templates.

In this paper, we introduce a technique for reducing the cost of utilizing new public SLA templates. In particular, after a new public SLA template is derived, users' SLA mappings submitted to the initial public SLA template are autonomically modified in order to be applicable to the new public SLA template. This approach improves the utility of market participants, since they may utilize new public templates that reflect their needs without any expenses. Furthermore, the initial public SLA templates can be deleted and replaced by the new templates, avoiding continuously

978-0-7695-4592-9/11 $26.00 © 2011 IEEE
DOI 10.1109/UCC.2011.38

increasing number of public templates on the market and making the market more efficient.

Besides autonomic creation of SLA mappings, in this paper, we introduce a new method for grouping similar user requirements that considers not only the structures of private SLA templates, i.e., definitions of SLA parameters, but the values of Service Level Objectives (SLOs) as well. SLOs are agreed values of SLA parameters representing obligations of the trading parties. With this approach, we additionally increase the utility of market participants and provide more advantageous options for creating product niches.

The main contributions of this paper are: (1) description of the autonomic SLA mapping creation approach; (2) grouping similar user requirements based on SLO values; (3) the formalization of a measure for evaluating the approach by determining the utility and cost to users; and (4) the evaluation of the approach using an experimental testbed.

The rest of the paper is organized as follows. Section II describes related work. Section III explains the adaptive SLA mapping approach and introduces methods for autonomic creation of SLA mappings. Methods for achieving cost-efficient utilization of public SLA templates are discussed in Section IV. Section V describes the simulation testbed, formalizes the utility and cost model, and presents the evaluation results. Section VI concludes the paper.

II. Related Work

Specifications of user requirements in distributed systems (Clouds and Grids) has been discussed by several research projects [5]–[7]. As reported in [8], most projects use SLA specifications based on WSLA and WS-Agreement, which lack support for economic attributes and negotiation protocols. To compensate these shortcomings, [5] introduce utilization of semantic Web technologies based on WSDL-S and OWL for enhancement of WS-Agreement specifications to achieve autonomic SLA matching. Similar to that, [6] and [7] suggest producing a unified QoS ontology applicable to the main scenarios, including ontology-based SLA formalization. However, they discuss only static SLAs and, therefore, do not consider market changes.

Several research projects have discussed the implementation of system resource markets [9]–[11]. GRACE [9] 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 [10] also considered open Grid markets. They identified several market requirements, such as allocative efficiency, budget-balance, truthfulness, and individual rationality [11]. However, they have not considered that a market can only function efficiently with a sufficiently large liquidity.

As reported by Kephart and Chess [12], the scientific community has in recent years focused on making distributed systems and supporting technologies 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 autonomic service management frameworks without explicitly taking economic methodologies into account [14]–[16]. On the other hand, research on autonomic systems focusing on economic methods and considerations, first proposed by [17], is in its early stage. For example, [18] propose mechanisms that are able to adaptively adjust their parameters based on the past behavior of participants. A self-organizing resource allocation mechanism in dynamic Application Layer Networks is proposed by [19]. They do not, however, consider issues such as the market adaptation depending on the available resources, which is a crucial element for potential autonomic marketplaces.

III. Towards Autonomic Cloud Markets

In our vision of Cloud markets, traders express their requirements and offers in form of *SLA templates*. They are documents comprising SLA parameters. Each SLA parameter is given by its *description*, stating basic parameter properties (e.g., parameter name), *metric*, representing a method of calculating the parameter, and *Service Level Objective* (SLO), i.e., contract's agreed value of the parameter. SLOs are in SLA templates defined by ranges of acceptable values. For example, an 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.

Consumers' requests and providers' offers are expressed by means of *private SLA templates*. On the other hand, *public SLA templates* are stored in publicly available, searchable SLA registries and represent groups of products traded on the market. For example, a public SLA template might be defined for medical applications. When a provider wants to offer a Cloud service on a market, he must first associate his service to a public SLA template, i.e., position his offer in one of the existing groups of products. This facilitates finding appropriate trading partners, enables better consumer support, and possibly increases providers' profit, due to the possibility of creating specialized product niches.

However, for the successful trading between two parties, SLA templates must be exactly the same in their structures in order to be understood by both sides. This implies that users' private SLA templates must be equal to the public SLA templates, which is often not the case. For this purpose, we utilize the *SLA mapping* technique [4]. SLA mappings are documents used to map the differences between two SLA templates. We differentiate between *ad-hoc* SLA mappings, which define a translation between a parameter existing in both SLA templates, and *future* SLA mappings, which define a wish for deleting an SLA parameter from a public SLA template or adding a new parameter that is supported by

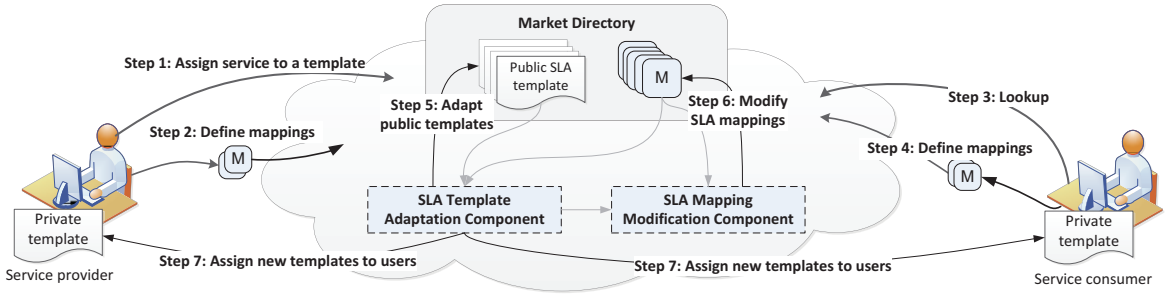

Figure 1. The SLA mapping approach

user's private template to a public SLA template. Properties mapped by ad-hoc mappings can range from very simple, such as incompatible element names (e.g., different names for the same SLA parameters) to very complex distinctions, such as differences in methods for calculating the parameter values (e.g., different units for expressing the same values).

A. The SLA Mapping Approach

Figure 1 depicts the market adaptation process. In the first step, a service provider assigns a service to a particular public SLA template. For example, if a provider offers a data-intensive surgical application for use, he might associate it with a public SLA template for medical applications. If the provider's private SLA template describing the service differs from the public template, he must define SLA mappings to bridge those differences (step 2). In the next step, a consumer looks for a Cloud service to use. He fetches a public SLA template and, if his private SLA template differs, he creates SLA mappings to map those differences (step 4).

At a certain point of time, the public SLA template adaptation process is started (step 5). This process is executed in several steps, as depicted in Figure 2. First, clustering algorithms are applied to group structurally similar user requirements based on submitted SLA mappings. Two SLA templates are similar by their structures if parameter definitions and metrics from the templates are similar, based on the distance function defined in Section IV. With this approach, the algorithm may update or even create new branches of existing public SLA templates. For example, when adapting a public template for 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, for each generated cluster, clustering algorithms are applied to create subgroups of requirements by differing in 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 done by applying learning methods (step 3 in Figure 2), which for each parameter from the initial public SLA template determine if its properties should be changed, or if a new parameter should be introduced, or an existing one deleted. After new templates have been created, they are published in the registry, and the old ones are deleted.

B. Autonomic SLA Mapping Adaptation

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 expenses for creating more SLA mappings to the new SLA templates, while the usage of the existing templates does not incur any additional cost. This results in a constant increase of number of public SLA templates, which may raise the expenses for the market, since the registry holds and maintains both old and new SLA templates. Therefore, a market must motivate users to utilize new public SLA templates, which is possible only without incurring them additional costs. To counteract this problem, in this paper, we investigate autonomic modification and creation of SLA mappings. Namely, we update the existing and possibly create new SLA mappings for users by utilizing the same rules used to transform the initial public SLA templates into the new public templates (step 6 in Figure 1). This approach dramatically reduces the cost for users and enables the market to delete the old public SLA templates, ensuring low cost of maintenance. The process of autonomic adaptation of SLA mappings is explained in detail in Section IV.

IV. Achieving Cost-Efficient Utilization of Public SLA Templates

In this section, we discuss the algorithms and methods implemented for grouping similar user requirements and generating new SLA templates. Besides, we explain the process of the autonomic creation of SLA mappings.

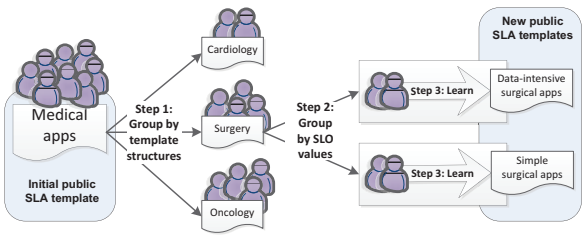

Figure 2. Adaptation of SLA templates

A. Public SLA Template Management

In the first step of the adaptation process, as mentioned in Section I, a clustering algorithm is applied to group similar requirements of market participants. User requirements are specified by private SLA templates, which can be reconstructed from the public SLA templates by applying users' SLA mappings. Therefore, it is possible to say that the clustering algorithm is applied to group users' private SLA templates. This process is executed in two steps. In the first step, as described in [4], submitted SLA mappings are analyzed and users' private SLA templates grouped so that one cluster contains those templates that are similar by their *structures*. In this paper, we introduce the second step, in which for each group of private SLA templates a clustering algorithm is applied to group templates into new clusters based on the *SLO values* contained by the templates.

For grouping users' private SLA templates we apply the *k*-means clustering algorithm. *k*-means partitions N data points into k disjoint subsets S_j containing N_j data points so as to minimize the sum-of-square criterion $\sum_{j=1}^{k}\sum_{n\in S_j}|x_n-\mu_j|^2$, where x_n is the nth data point and μ_j is the geometric centroid of the data points in S_j [20]. 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. In our discussion of grouping user requirements, 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. In this paper, k-means creates $\sqrt{N/2}$ clusters, while in our future work we plan to introduce a heuristic method for determining the number k based on a measure of market liquidity.

In order to be able to utilize clustering algorithms, we must determine (dis)similarity of two data points, i.e., two SLA templates. Furthermore, based on the submitted SLA mappings, we must define a method for computing a cluster centroid, i.e., a new public SLA template for a group of users. In the following, we explain our approaches to counteract the two issues.

1) Computing Distance Between SLA Templates: We define two methods for computing distance between SLA templates: one in respect to **their structures**, and one in respect to **their SLO values**.

The distance between the structures of SLA templates is expressed as a number of differences between parameter properties of SLA templates. For two SLA templates T_1 and T_2, the distance between their structures S in respect to an SLA parameter p and its properties (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 two SLA templates in respect to their structures is calculated as sum of distances between all SLA parameters contained by at least one of the templates.

When calculating the distance between two templates in respect to their SLO values, we must consider that the SLO values are in SLA templates represented by ranges of real numbers, as explained in Section I. In order to compute the distance between such ranges, we utilize Hausdorff metric [21], which defines this value as a maximum distance of one range to the nearest point in the other range. In detail, if X and Y are two non-empty subsets of a metric space (M,d), their Hausdorff distance $d_H(X,Y)$ is defined by

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

where sup represents the supremum and inf the infimum. Let values of an SLO V from an SLA template T_1 be $V_{T_1}=[x_1,y_1]$ and SLO value from an SLA template T_2 be $V_{T_2}=[x_2,y_2]$. For any two bounded ranges V_{T_1} and V_{T_2} it is simple to show that their Hausdorff distance is

$$d_H(V_{T_1},V_{T_2})=max(|x_1-x_2|,|y_1-y_2|) \quad (2)$$

The total distance between two SLA templates in respect to their SLO values is calculated by computing distances between values defined in at least one of the templates and adding them up. In order to be mutually comparable, they are fitted into a range $[0,1]$ before summing. This is done by applying the following equation:

$$d_V(T_1,T_2)=\frac{d_H(V_{T_1},V_{T_2})-min(d_{H,V})}{max(d_{H,V})-min(d_{H,V})} \quad (3)$$

where $min(d_{H,V})$ is the minimum distance between any two SLA templates in respect to the values of the SLO V and $max(d_{H,V})$ its maximum. 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 from the public template before computing the distance.

2) Adapting SLA Templates: For each computed cluster of users' private SLA templates, we must be able to compute a cluster centroid, i.e., derive a new public template from an initial public SLA template depending on the needs of the group of users. For this purpose we define three learning methods. These methods analyze submitted SLA mappings and for each SLA parameter determine whether the current parameter description and metric should be changed, as well as if a new parameter should be added or an existing one deleted. As in [4], we utilize the following methods:

a) *maximum method*, which selects a candidate with the highest number of SLA mappings (maximum candidate);
b) *threshold method*, where the value gets chosen if it has the highest count and is used more than the given threshold (in our simulations fixed at 60%); and

c) *significant-change method*, where a parameter property is only changed if the percentage difference between the highest count value and the current public template value exceeds a given threshold, which we fix at $\sigma_T > 15\%$.

After adapting template structures, SLO values are converted to possibly changed SLO units and updated in order to reflect user requirements. An SLO value is changed so that the lower limit of the range is set to the minimal value and the upper limit to the maximal value contained by any private SLA template in the given group of users.

B. Autonomic SLA Mapping Modification

In order to reduce the cost of creating new SLA mappings for users and therefore make the market more attractive, once a new public SLA template has been introduced to the market, users' SLA mappings are redefined so that they can be used for the newly created public SLA templates in future processes. In our discussion of the algorithm for autonomic SLA mapping modification, we observe a user's private SLA template T_u, an initial public SLA template T_i, and a newly generated public SLA template T_n. An SLA parameter α is in an SLA template T defined by its description $D_T(\alpha)$, metric $F_T(\alpha)$, and SLO value $SLO_T(\alpha)$, noted $[D_T(\alpha), F_T(\alpha), SLO_T(\alpha)]$. An ad-hoc SLA mapping between descriptions of an SLA parameter α in SLA templates T_1 and T_2 is noted as $D_{T_1}(\alpha) \leftrightarrow D_{T_2}(\alpha)$. Finally, we define function χ to determine if T contains α:

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

The algorithm iterates through all SLA mappings applied to transform T_i into T_n, and for each mapping executes one of the possible actions, as depicted in Figure 3. Since a mapping for a parameter α only exists if there is a difference in its definition in two SLA templates, we assume that α exists in at least T_i or T_n and that its properties differ in those templates. In cases where this does not hold, existing user's SLA mappings for this parameter are kept intact.

In case α exists in all three SLA templates (line 1), one of the following actions is executed:

1) If a parameter property did not differ in T_i and T_u, but it changed in T_n, a new ad-hoc mapping is created to map the newly created difference (lines 2-3).
2) If a parameter property differs in all three templates, a new ad-hoc SLA mapping is created (lines 4-5) that is a combination of the existing mappings so that the output of one becomes the input for another mapping, as illustrated in Figure 4.
3) If a parameter property does not differ in T_u and T_n, existing ad-hoc SLA mapping is deleted (lines 6-7).

In case α was deleted from T_i, but it is needed by the user, the algorithm creates a new adding wish (lines 9-10) and deletes possibly existing ad-hoc mapping for the deleted parameter (lines 11-12). On the other hand, if it was deleted from T_i, but the user does not need it, the existing deleting wish is removed (lines 14-15). If a new parameter is introduced in T_n (line 16), the algorithm deletes existing adding wish if it recognizes the equivalent from T_u (lines 17-18), or it creates a deleting wish for the newly added parameter (lines 19-20). Note that the recognition of the newly added parameter's equivalent in the user's private SLA template is possible only if the user's adding wish is exactly the same as the one applied to modify the initial public SLA template. Finally, if the parameter properties changed in T_i and T_n, but the user does not utilize the parameter, the existing deleting wish is kept (lines 22-23). Note that in Figure 3 we compare only parameter descriptions, while the implementation also considers their metrics. This is done by simply replacing D_T with F_T in the given algorithm.

1: **if** $\chi_{T_n}(\alpha) = \chi_{T_i}(\alpha) = \chi_{T_u}(\alpha) = true$ **then**
2: **if** $D_{T_i}(\alpha) \neq D_{T_n}(\alpha) \wedge D_{T_i}(\alpha) = D_{T_u}(\alpha)$ **then**
3: create $D_{T_n}(\alpha) \leftrightarrow D_{T_u}(\alpha)$
4: **else if** $D_{T_n}(\alpha) \neq D_{T_i}(\alpha) \wedge D_{T_n}(\alpha) \neq D_{T_u}(\alpha) \wedge D_{T_i}(\alpha) \neq D_{T_u}(\alpha)$ **then**
5: combine $D_{T_n}(\alpha) \leftrightarrow D_{T_i}(\alpha)$ and $D_{T_i}(\alpha) \leftrightarrow D_{T_u}(\alpha)$
6: **else if** $D_{T_n}(\alpha) \neq D_{T_i}(\alpha) \wedge D_{T_n}(\alpha) = D_{T_u}(\alpha)$ **then**
7: delete ad-hoc mapping $D_{T_i}(\alpha) \leftrightarrow D_{T_u}(\alpha)$
8: **end if**
9: **else if** $\chi_{T_n}(\alpha) = false \wedge \chi_{T_i}(\alpha) = \chi_{T_u}(\alpha) = true$ **then**
10: create adding wish $[D_{T_u}(\alpha), F_{T_u}(\alpha), SLO_{T_u}(\alpha)]$
11: **if** $D_{T_i}(\alpha) \neq D_{T_u}(\alpha)$ **then**
12: delete existing ad-hoc mapping $D_{T_i}(\alpha) \leftrightarrow D_{T_u}(\alpha)$
13: **end if**
14: **else if** $\chi_{T_i}(\alpha) = true \wedge \chi_{T_n}(\alpha) = \chi_{T_u}(\alpha) = false$ **then**
15: delete existing deleting wish for the parameter α
16: **else if** $\chi_{T_i}(\alpha) = false \wedge \chi_{T_n}(\alpha) = true$ **then**
17: **if** $\exists \mu : \chi_{T_u}(\mu) = true \wedge \chi_{T_i}(\mu) = false \wedge D_{T_u}(\mu) = D_{T_n}(\alpha) \wedge F_{T_u}(\mu) = F_{T_n}(\alpha)$ **then**
18: delete existing adding wish for the parameter μ
19: **else**
20: create deleting wish for the parameter α
21: **end if**
22: **else if** $\chi_{T_i}(\alpha) = \chi_{T_n}(\alpha) = true \wedge \chi_{T_u}(\alpha) = false$ **then**
23: keep existing deleting wish for the parameter α
24: **end if**

Figure 3. Algorithm for autonomic SLA mapping modification

Ideally, by autonomically modifying users' SLA mappings, we eliminate the cost of creating new SLA mappings for the users. However, this is not always the case. Most of the created SLA mappings can be directly used without causing any concern. However, in case an important parameter gets deleted from the initial template, although the algorithm

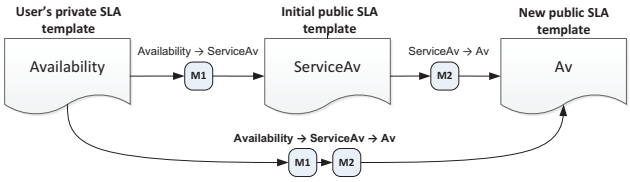

Figure 4. Building an SLA mapping as a combination of two mappings

Table I
USER'S UTILITY PER SINGLE SLA PARAMETER

Utility	Cause
0	The parameter does not exist in one of the SLA templates
A	Both parameter properties differ in the SLA templates
B	Only one parameter differs in the SLA templates
C	SLA templates do not differ with respect to the parameter

Table II
USER'S COST PER SINGLE SLA PARAMETER

Cost	Cause
0	User needs no new SLA mappings
D	User must create new ad-hoc SLA mapping(s)
E	User must create a new future SLA mapping
F	User must rectify autonomically created SLA mappings

creates an adding wish to substitute this loss, a user might not be satisfied by using a new version of the template, since it might not reflect his requirements in an acceptable extent. Therefore, in case of autonomic creation of adding wishes, we must warn the user about the possible consequences. Furthermore, in case a new SLA parameter has been added into the public template, the algorithm cannot determine whether the user needs the parameter or not. Namely, in case a user's private SLA template contains the same parameter, but with a different property, we cannot recognize that those are equivalents, due to the lack of semantic description of SLA parameters. In this paper, we solve this problem by keeping an adding wish to add the parameter from the private SLA template and creating a deleting wish to delete a newly introduced parameter from the new public SLA template. Furthermore, the market notifies the user and warns him about the possible consequences of the adaptation process.

V. EVALUATION

A. Utility and Cost Model

To assess the benefits of our approach, we define a utility and cost model. The model quantifies the advantages of the approach, as well as the costs it incurs. In this paper, we modify the model mathematically formalized in [4].

The utility and cost functions define utility and cost for a single user per one SLA parameter and they are calculated for all parameters contained by user's private SLA template or by the newly created public SLA template associated to the user's service. Overall utility and overall cost are calculated by summing utility and cost for all users and for all SLA parameters. The difference between the overall utility and overall cost determines the overall net utility of the users and is used for the evaluation of the approach.

Simplified representation of the utility function is given in Table I. As stated, a user has no utility for an SLA parameter if the parameter does not exist in user's private SLA template or associated public SLA template, since the user cannot utilize such parameter. If the parameter is contained by both SLA templates, but with different properties (parameter description and metrics), the user has utility A. If only one of the properties differs in the templates, the user has utility B. Finally, if the parameter does not differ in the two SLA templates, the user has the maximum utility C.

Table III
VALUES FOR THE UTILITY AND COST MODEL USED FOR SIMULATION

Variable	A	B	C	D	E	F
Value	1	2	3	1	2	5

The cost function depicted in Table II states that a user has no cost for an SLA parameter if he does not have to create any new SLA mappings before utilizing the parameter. This occurs when parameter properties are the same in the user's private template and in the new public SLA template, or if they differ, but SLA mappings bridging the differences already exist. The latter might happen if the new public SLA template does not differ from the initial public template in respect to the parameter, so that the user has already created the mappings in the last iteration, or if it changed, but the mappings were autonomically created for the user. If a user must create one or two ad-hoc SLA mappings, due to the differences between one or both parameter properties, user has the cost D. If he must create a future SLA mapping, i.e., an adding or a deleting wish for a missing SLA parameter, the cost is equal to E. Finally, the user has the cost F if he must rectify his SLA mappings. This might happen if the new public SLA template has a new SLA parameter existing in the user's private SLA template, but with a different property. In this special case, the autonomic SLA mapping manager cannot recognize that the user's parameter is the same as the newly added parameter in the public SLA parameter and instead of creating ad-hoc SLA mappings, it creates a deleting wish for the newly added SLA parameter and an adding wish for the parameter existing in the user's private SLA template. As mentioned in Section IV, this is the only scenario in which autonomic modification of user's SLA parameters fails to satisfy user's needs.

Note that the more similar SLA templates are, the more a user is attracted to the market and gets higher utility. Considering the cost, we assume that creating a future SLA mapping incurs more cost than creating an ad-hoc SLA mapping, due to the mapping complexity. Therefore, although the values for the utility and cost functions are not strictly defined, considering effort or benefit, it should hold that $0 \leq A \leq B \leq C$ and $0 \leq D \leq E \leq F$. In our simulations, we used the values depicted in Table III.

B. Simulation Environment

For the simulation purposes, we designed a framework for automated management of SLA mappings and generation of SLA templates. The framework is built out of three main components, implemented in Java: market repository, frontend services, and market modification component.

Market repository is a publicly available and searchable directory used for storing and managing public SLA tem-

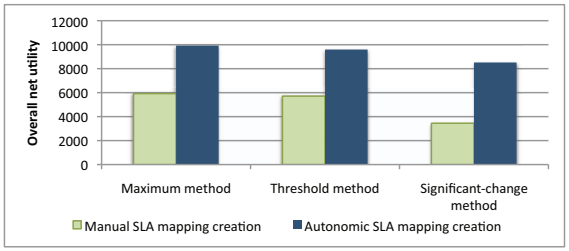

Figure 5. Simulation results: overall net utility

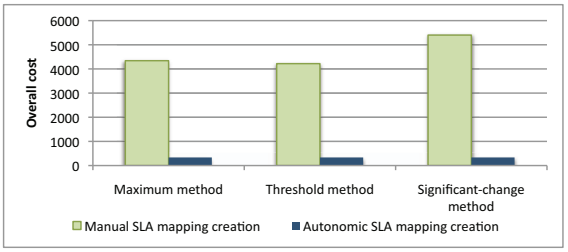

Figure 6. Simulation results: overall cost

plates and SLA mappings. **Frontend services** are market services publicly available to the users. The services are used for administration (e.g., creating SLA templates), accounting (e.g., creating consumer accounts), querying (e.g., finding an appropriate SLA template), and management of SLA mappings (e.g., defining mappings for a user). **Market modification** component includes mechanisms for autonomic creation and management of public SLA templates and adaptation of users' SLA mappings.

The simulation is conducted as described in Section III. After new SLA templates have been derived and users' SLA mappings adapted, the overall net utility is measured to evaluate the process. In our simulation, the initial public SLA template is static and 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 for each SLA parameter users can choose between 5 predefined parameter descriptions and 4 metrics. Table IV summarize the simulation settings.

Table IV
SIMULATION SETTINGS

Parameter	Value
Number of service providers	1
Number of service consumers	500
Number of initial public SLA templates	1
Number of parameters in initial public SLA template	8
Number of parameters in private SLA templates	$\leq$11
Size of the set of possible descriptions per SLA parameter	5
Size of the set of possible metrics per SLA parameter	4

C. Evaluation Results

In this section, we measure and compare the overall net utility achieved by both the approach introduced in this paper and the one presented in [4], where the SLA mappings were not autonomically created after the adaptation of the public SLA templates, but manually by the users.

Figure 5 presents the overall net utility achieved by the two approaches for each of the learning methods. As depicted, by autonomically modifying existing SLA mappings, the overall net utility is significantly higher than when the users must manually create new mappings. Note that the overall utility does not differ in the two approaches, since the newly created public templates are equal. Therefore, the difference in overall net utility is caused only by the reduction of the cost for creating SLA mappings.

The difference between costs incurred by the approaches is depicted in Figure 6. Since in our new approach the users are not required to create any new SLA mappings, the overall cost for users is reduced almost to 0. The cost still exists since not all of the autonomically created SLA mappings fully satisfy users' needs. In particular, as explained in the definition of the utility and cost model, the adaptation algorithm might not correctly recognize a newly added parameter's equivalent in a user's private SLA template. This, of course, incurs cost, since the user must rectify created mappings. However, note that the new SLA parameters are not added in public templates on a regular basis, but rather seldom, in case of a large change in user requirements. Therefore, the overall cost for users is in most cases of the adaptation process equal to 0.

It is important to note that the cost for creating SLA mappings has not vanished, but shifted from the users to the market. However, the cost for creating SLA mappings is in this case negligible, since they are adapted autonomically instead of manually and therefore do not require human interaction. Additionally, since the mappings are immediately updated, initial public templates can be deleted and replaced by newly created SLA templates. This was not possible in the previous approach, due to the necessity to create new mappings before utilizing newly created public templates. This was sometimes hard to achieve, since the users preferred keeping the old public templates so to avoid additional cost of creating new SLA mappings. By dynamically replacing public SLA templates, the cost for maintenance and storage of public SLA templates is reduced.

As depicted in Figure 5, the maximum method achieves the highest rate of overall net utility, although it incurs more cost than the threshold method, as depicted in Figure 6. This is due to more frequent changes of parameter properties, which increases the utility, but also the cost, since every change requires a new SLA mapping.

VI. CONCLUSION AND FUTURE WORK

In this paper, we introduced an approach for cost-efficient utilization of public SLA templates in autonomic Cloud markets. Namely, we presented a method for autonomic modification and creation of SLA mappings in order to reduce the cost of creating SLA mappings for market participants, allowing them to utilize new public SLA templates

representing the current market trends without any effort. Furthermore, we investigated autonomic creation and management of public SLA templates both by their structures and their SLO values. Our evaluation model based on the simulation framework showed that our approach increases the overall net utility of traders and market in general and lowers the cost of market maintenance.

In our future work, we will investigate the possibilities of reducing the cost of creating SLA mappings to the initial public SLA templates. We will do this by introducing a knowledge component managing the history of previous private SLA templates and SLA mappings, and apply machine learning methods to advise users of possible mappings to create. Furthermore, we will explore the methods for measuring market liquidity and adapt public SLA templates so as to increase the liquidity to its maximum point.

ACKNOWLEDGMENT

The authors would like to thank Jörn Altmann for fruitful discussions. This work was supported by the Vienna Science and Technology Fund under the grant agreement ICT08-018 Foundations of Self-Governing ICT Infrastructures (FoSII).

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, and M. Zaharia, "Above the clouds: A Berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep., February 2009.

[2] M. Risch, I. Brandic, and J. Altmann, "Using SLA mapping to increase market liquidity," in *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, ser. Lecture Notes in Computer Science. Springer, 2010, vol. 6275, pp. 238–247.

[3] M. Maurer, V. C. Emeakaroha, M. Risch, I. Brandic, and J. Altmann, "Cost and benefit of the SLA mapping approach for defining standardized goods in cloud computing markets," in *International Conference on Utility and Cloud Computing (UCC 2010) in conjunction with the International Conference on Advanced Computing (ICoAC 2010)*, 2010.

[4] I. Breskovic, M. Maurer, I. Brandic, V. Emeakaroha, and J. Altmann, "Towards autonomic market management in cloud computing infrastructures," in *International Conference on Cloud Computing and Services Science. CLOSER*, 2011.

[5] N. Oldham and K. Verma, "Semantic WS-agreement partner selection," in *15th international conference on World Wide Web. WWW '06*. ACM Press, 2006, pp. 697–706.

[6] G. Dobson and A. Sanchez-Macian, "Towards unified QoS/SLA ontologies," in *Services Computing Workshops, 2006. SCW '06. IEEE*, 2006, pp. 169 –174.

[7] L. Green, "Service level agreements: an ontological approach," in *8th international conference on Electronic commerce*, ser. ICEC '06. ACM, 2006, pp. 185–194.

[8] P. Karänke and S. Kirn, "Service level agreements: An evaluation from a business application perspective," in *eChallenges e-2007*. IEEE-CS Press, 2007, pp. 104–111.

[9] R. Buyya, D. Abramson, and J. Giddy, "A case for economy grid architecture for service oriented grid computing," *Parallel and Distributed Processing Symposium*, vol. 2, 2001.

[10] J. Nimis, A. Anandasivam, N. Borissov, G. Smith, D. Neumann, N. Wirstrm, E. Rosenberg, and M. Villa, "SORMA: Business cases for an open grid market: Concept and implementation," in *Grid Economics and Business Models*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, vol. 5206, pp. 173–184.

[11] D. Neumann, J. Stösser, and C. Weinhardt, "Bridging the adoption gap-developing a roadmap for trading in grids," *Electronic Markets*, vol. 18, pp. 65–74, February 2008.

[12] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, pp. 41–50, January 2003.

[13] I. Brandic, D. Music, and 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*. Barcelona, Spain: ACM, 2009, pp. 1–8.

[14] Y. Cheng, A. Leon-Garcia, and I. Foster, "Toward an autonomic service management framework: A holistic vision of SOA, AON, and autonomic computing," *Communications Magazine, IEEE*, vol. 46, no. 5, pp. 138 –146, may 2008.

[15] W. H. Oyenan and S. A. Deloach, "Towards a systematic approach for designing autonomic systems," *Web Intelligence and Agent Systems*, vol. 8, pp. 79–97, January 2010.

[16] C. Lee and J. Suzuki, "An autonomic adaptation mechanism for decentralized grid applications," in *Consumer Communications and Networking Conference, 2006. CCNC 2006. 3rd IEEE*, vol. 1, jan. 2006, pp. 583 – 589.

[17] G. Cheliotis and C. Kenyon, "Autonomic economics," in *E-Commerce, 2003. CEC 2003. IEEE International Conference on*, june 2003, pp. 120 – 127.

[18] D. Pardoe, P. Stone, M. Saar-Tsechansky, and K. Tomak, "Adaptive mechanism design: a metalearning approach," in *Proceedings of the 8th international conference on Electronic commerce*, ser. ICEC '06. ACM, 2006, pp. 92–102.

[19] W. Streitberger and T. Eymann, "A simulation of an economic, self-organising resource allocation approach for application layer networks," *Computer Networks*, vol. 53, pp. 1760–1770, July 2009.

[20] 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.

[21] G. Rote, "Computing the minimum hausdorff distance between two point sets on a line under translation," *Information Processing Letters*, vol. 38, pp. 123–127, 1991.