A Rule-Based Method for Comparison of SLAs in Service-Oriented Computing

Laima Paliulionienė *laima.paliulioniene@mii.vu.lt*
Vilnius University Institute of Mathematics and Informatics
Vilnius, Lithuania

Abstract

Service selection based on non-functional properties (NFP) of services is one of the most important and challenging task in service-oriented computing. NFP are usually described in service-level agreements (SLA). Therefore, the problem of comparison of SLAs arises naturally if there is more than one service with the same desired functionality. SLAs are usually written in a special XML-based format (WS-Agreement, WSLA, RBSLA, etc.). They describe some rules of using the service at the proper level, set the metrics by which that service is measured, and the remedies or penalties in case the agreed-upon levels are not achieved. The paper proposes a method to find differences between SLAs using rule-based knowledge representation and first-order logic-based derivation.
Keywords: Service-oriented Computing, Service-level Agreement, Quality of Service, Rule-based Knowledge Representation, Model Generating.

1. Introduction

Service-oriented computing is increasingly gaining ground in the developing of enterprise applications and sets a particular type of relationships between and inside business entities. This inherently raises an issue of contracting relationships, as services in real life are subjects for negotiation and contracting.

The term "contract" has originated from law and means an agreement with legal obligation. However, in the service-oriented architecture (SOA), the definition of the service contract achieves mostly technical rather than legal spirit. In case of Web Services, that are the most common implementation of SOA services, the contract usually means service description documents – WSDL definition, XSD schema, and policy [4]. It should be noted that mostly technical understanding follows the traditions of Bertrand's Meyers's "design by contract" that was first introduced in 1986, in connection with the creating of the object-oriented programming language Eiffel. Beside such contracts, there are other types of agreements in SOA. The most popular is a Service Level Agreement (SLA). SLAs describe rules of using the service at the level expected by a customer from a provider, define their obligations and rights, set metrics to measure quality of service (QoS) (for example, mean time between failures, mean time to repair, various data rates, throughput, etc.), and the remedies or penalties in case the agreed-upon levels are not achieved. SLAs are usually written in a special XML-based format (WS-Agreement, WSLA, RBSLA, etc.).

The problem of SLAs analysis arises naturally if there is more than one service with the same desired functionality and it is necessary to select one of them. To this aim, one needs to compare non-functional properties of the services. The comparison of SLAs is also useful in negotiations, as it can help to put a negotiation offer to change some item of a SLA proposed by a provider, on the grounds of SLAs from other providers. Moreover, the detection of differences is useful in the managing changes in SLAs.

The paper proposes a method to find differences between SLAs using rule-based knowledge representation and first-order logic-based derivation.

The remainder of this work is organized as follows: in Section 2, related work is discussed; Section 3 presents the proposed method for SLAs comparison, including

definitions of difference between SLAs, model generating, algorithms, and proofs of related theorems; Section 4 contains conclusions of the work.

2. Related Work

WSLA [8] and WS-Agreement [2] are the most widely accepted approaches to describe SLAs. However, they have limited expressiveness to represent rule sets and complex conditionals. In [11, 12], a declarative rule-based knowledge representation framework for SLA and policy representation and management is proposed by introducing ContractLog and RBSLA languages. The ContractLog framework provides a declarative rule language based on logic programming and explicitly expressing contractual logic. The ContractLog knowledge representation uses an extended ISO Prolog related scripting syntax. RBSLA (Rule-Based Service Level Agreement) presents XML syntax for ContractLog knowledge representation. It adapts and extends the XML-based rule standard RuleML to the needs of the SLA domain. According to [6], SLAs that are written in WS-Agreement or similar language can also be transformed into corresponding sets of logical rules.

There are two main trends in service selection approaches: optimization-based approaches and negotiation-based approaches [10]. In the first approach, services are ranked by a value of utility on the ground of some criteria. The value of utility is calculated encompassing all QoS attributes. The ranking of services allows to compare them by their suitability for a customer. No rule-based derivation is made in this case, only numerical values of attributes are considered. In optimization-based approach, these values are assumed to be undiscussable. In contrast, negotiation-based approaches assume the possibility to change SLAs through information exchange and compromises between a customer and provider. A good review of existing service selection approaches is presented in [10].

Most papers on SLA comparison are focused on the computable parts of SLAs (e.g., [1], [3], [5]). As mentioned previously, they propose methods of ranking services on the basis of quantitative attributes (e.g., response time, availability, accuracy, and cost) and weights of the attributes, indicating their relative importance for a service customer. Our work differs from others in that we consider rule-based representation of SLAs and investigate how to compare logical consequences (not necessary numeric values) that can be derived from the SLAs. Our method can help to analyze and compare rules that cannot be expressed in WS-Agreement or similar language, but can be expressed in rule-based language. At the same time, mainly all SLAs can be transformed into rule-based form, and therefore they become objects for the proposed rule-based method.

In order to make possible the comparison of SLAs, it is important to have some common standards or ontology for SLAs. According to the European Commission report [7], there is a serious lack of standards for different elements and parts of the SLA lifecycle. In particular, for the quantitative and qualitative comparison of SLAs, it is expedient to develop an SLA Reference model. At present, SLA attributes are not universally standardized. SLA templates are often used to define the structure of an SLA, the names of the service attributes, and the attribute values [9]. As there can be a variety of different public and private SLA templates, a problem of the template matching arises. Service Measurement Index (SMI) was proposed by the Cloud Service Measurement Index Consortium (CSMIC) [3], [5] for measuring and comparing services. It ranges QoS attributes in the hierarchical structure. Appropriate attribute names of different services are not required to be unified, and their matching is a separate task. SLA meta-models that are proposed in [1] also seek to generalize and standardize SLA attributes. So, though it is so far unrealistic to require the same ontology from different service providers, SLAs can be transformed to some common standards in order to be comparable. The selection of a particular method of such standardization (SLA Reference model, template matching, meta-models, or other) is beyond the scope of this paper.

3. Method of SLAs Comparison

3.1. Basic Principles

When we have the task of the SLAs comparison, we assume that the SLAs are standardized and share the same ontology, as stated in Section 2.

In the proposed method, we also assume that SLAs are formalized as logic programs, with facts and rules, and are stored in knowledge bases. Thus, the task of comparison of SLAs is partly transformed into the task of comparison of appropriate knowledge bases.

Rules in the knowledge bases have the form *head←body*. Facts are rules without a body, with a head only. The head (also called the *consequent*) is an atom (a positive literal), and the body (also called the *antecedent*) is a conjunction of literals, that can be positive and/or negative. This is Prolog-like representation.

ContractLog has the similar knowledge representation. For example, the requirement "between 0 and 4 a.m., the service availability will be measured every 10 minutes, if maintenance is being performed" can be formalized as follows (untyped formalization is shown for simplicity; the symbol "`:-`" is used instead of the arrow "←") [11]:

```
schedule(maintenance, Service) :-
sysTime(datetime(Y,M,D,H,Min,S)),
lessequ(datetime(Y,M,D,H,Min,S), datetime(Y,M,D,4,0,0)),
interval(timespan(0,0,10,0), datetime(Y,M,D,H,Min,S)),
service(Service), maintenance(Service).
```

As it was stated above, SLAs that are written in other languages (including the most popular WS-Agreement) can be transformed into corresponding sets of logical rules. Therefore, our comparison method is suitable not only for initially rule-based RBSLA/ContractLog.

We specify the task of the comparison of SLAs as the detection of differences between them. We define the difference between SLAs as the difference between conclusions derived from possible situations. The set of conclusions is actually a model of the logic program. The comparison of SLAs is considered as the comparison of models generated from the SLAs. The proposed model generating algorithms is presented in the next section.

3.2. Model Generating Algorithm

The comparison of SLAs is considered as the comparison of models generated from the SLAs. We use the model generating algorithm that implements negation as failure. We will denote $KB \vdash_M N$ to state that the model N can be generated from the knowledge base KB using this algorithm. According to our algorithm, we firstly add facts to the model, and after that we add conclusions derived from the facts by using rule chains. Let's describe it formally.

Model generating algorithm. The initial model is an empty set. The model is generated as follows:

1. *Horn clauses*. Let the knowledge base has a rule $C \leftarrow A$, where A is a set (possibly the empty set) of positive literals, and C is a positive literal (that is, the rule is a Horn clause). Let there exists a ground substitution σ such that $A\sigma$ is true in the so far generated model N, and $C\sigma$ does not belong to N. Then the model N is to be supplemented with $C\sigma$. As it is known, a ground substitution in first-order logic is a total mapping σ: V → T from variables to variable-free terms.

Example 1. Let us look at the model generating in the simplest case, when there are no variables. Let the knowledge base consists of the following rules and facts:

1) $e \leftarrow a, d$. 3) $c \leftarrow a$. 5) $h \leftarrow g$. 7) b.
2) $f \leftarrow b, e$. 4) $d \leftarrow b$. 6) a.

The model of the knowledge base will be generated as follows:

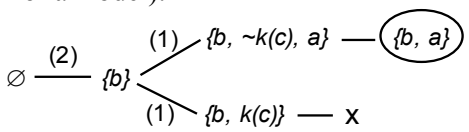
$$\varnothing \xrightarrow{(6)} \{a\} \xrightarrow{(7)} \{a,b\} \xrightarrow{(3)} \{a,b,c\} \xrightarrow{(4)} \{a,b,c,d\} \xrightarrow{(1)} \{a,b,c,d,e\} \xrightarrow{(2)} \{a,b,c,d,e,f\}$$

2. *Non-Horn clauses*. Let the knowledge base has a rule $C \leftarrow A$, *not* P_1, ..., *not* P_n (n>0), where A is a set (possibly the empty set) of positive literals and P_1, ..., P_n are positive literals (that is, the rule is non-Horn clause). Let there exists substitution σ such that $A\sigma$ is true in so far created model N, and $C\sigma$ does not belong to this model. Then N is to be supplemented in such a way that two models are given with special modal operator k: model $N \cup \{C\sigma, \sim k(P_1\sigma), ..., \sim k(P_n\sigma)\}$ and model $N \cup \{k(P_1\sigma, \ldots; P_n\sigma)\}$. Here $\sim k(P)$ means that it is expected that the model will not have the fact P, whereas $k(P_1; \ldots; P_n)$ means that it is expected that the model will have at least one of facts $P_1, \ldots, P_n$. Later we will eliminate models that contain both $\sim k(P)$ and P. If a model has $k(P_1; \ldots; P_i; \ldots; P_n)$ and some P_i (i=1,…,n), then $k(P_1; \ldots; P_i; \ldots; P_n)$ will be removed from the model. After finishing the generating of models, we will remove $\sim k(P)$ from models that does not have P; and we will eliminate models that have $k(P_1; \ldots; P_n)$, but do not have any P_i (i=1,…, n, n>0),. Therefore, final models will not have any facts with operator k.

Unlike the Horn clause case, in the case of non-Horn clauses we will receive not necessary only one model. It is possible to get several models or no model at all. Let us look again at the model generating in the simplest case, when there are no variables.

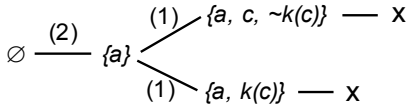
Example 2. The knowledge base below has only one model {*b*, *a*} that is generated in the following way (x means the elimination of a model):

1) $a \leftarrow b$, not c.
2) b.



Example 3. The knowledge base that is presented below has no models. The model generating is pictured on the right.

1) $c \leftarrow a$, not c.
2) a.



Example 4. Let the knowledge base contain two rules: 1) $a \leftarrow$ not c. 2) $c \leftarrow$ not a. This knowledge base will have two models – {a} and {c}.

We will consider only knowledge bases that satisfy the following restrictions:

1. *Bound variables*. All variables in rule consequents and negative literals of antecedents are also used in positive literals of the antecedent.

2. *Exactly one model*. The generated model is only one. This restriction is appropriate in the domain of SLA regulations, because every situation should imply concrete conclusions.

3. *Finite model*. The generated model is finite.

ContractLog/RBSLA has the possibility to use typed logic constructs, the scoped negation as failure, and some other additional features [12]. In the paper, we do not consider these cases.

3.3. Definition of Difference

Let us define formally what we call the difference between knowledge bases.

Definition 1. We say that there is the *internal difference* between knowledge bases KB_1 and KB_2, if $KB_1 \vdash_M R_1$, $KB_2 \vdash_M R_2$, and sets R_1 and R_2 are not equal. We define an *internal difference set* as $DiffInt(KB_1, KB_2) = (R_1 \cup R_2) \setminus (R_1 \cap R_2)$.

Definition 2. Let S be a set of facts describing a situation. We say that knowledge bases KB_1 and KB_2 are *different in situation S*, if $KB_1 \cup S \vdash_M R_1$, $KB_2 \cup S \vdash_M R_2$, and sets R_1 and R_2 are not equal. We define a *difference set in a situation S* as $Diff(S, KB_1, KB_2) = (R_1 \cup R_2) \setminus ((R_1 \cap R_2) \cup DiffInt(KB_1, KB_2))$. According to this definition, elements of the internal difference set are not included in the difference set.

Definition 3. Knowledge bases are *different* if there is internal difference between them or they are different in some situation.

3.4. Elementary Test Situations

In order to find all differences between knowledge bases, we need to find their internal difference, all situations that the knowledge bases are different in, and their difference sets in these situations. There can be a lot of possible situations. If function symbols are used in terms, then there can be the infinite number of situations. It is not possible to check all off them. We will show that instead of the checking of all possible situations it is sufficient to check test situations that we will define below. We need some additional definitions to this aim.

Definition 4. A substitution instance of a formula without negative literals is called an *elementary substitution instance*, if the substitution replaces all variables by variable-free terms (constants), and:

1) these constants have no functional symbols,

2) these constants do not belong to the Herband universe of the knowledge base (i.e. they are not used in the knowledge base),

3) different variables are replaced by different constants.

Definition 5. Let KB_1 and KB_2 be two knowledge bases to be compared. Let KB_1 contains a rule $C \leftarrow A$, *not* P_1, ..., *not* P_n ($n \geq 0$, A consists of atoms or is the empty set, P_1, ..., P_n are atoms). A set of facts E is called an *elementary test situation*, if one of the following conditions is true:

1) $E = A\theta$, where $A\theta$ is an elementary substitution instance of A (if $A=\varnothing$, then this condition is not considered),

2) $A\theta \subset E$, where $A\theta$ is an elementary substitution instance of A, and the knowledge base KB_2 contains at least one rule $D \leftarrow B$, *not* R_1, ..., *not* R_m ($m>0$, B consists of atoms or is the empty set, R_1, ..., R_m are atoms) such that $B\theta \setminus V_2 \subseteq A\theta$, where V_2 is the set of internal facts of KB_2, and there exists R_i ($1 \leq i \leq m$) such that $R_i\theta \in E$.

This definition means that if the positive part of the antecedent of a rule is a subset of a positive part of the antecedent of a rule from another knowledge base, then facts from the negative part of the antecedent must be also considered in constructing elementary test situations.

We propose the following algorithm for the construction of an elementary test situation from a knowledge base rule:

1. Construct the elementary substitution instance of the positive antecedent part of a rule in the following way:

1.1. replace every variable with a constant, using the variable name. If variables start from a capital letter like in Prolog, then the constant can be obtained just by replacing capital letters with the same small (lower-case) letters (for example, a variable X can be replaced with a constant x).

1.2. if the obtained constant belongs to the Herbrand universe of KB_1 or KB_2, we need to change its name in order to obtain a distinct constant. To this aim, its name can be appended by a symbol, for example, "1" (in this case, the constant *abc* will be changed to *abc1*). If the obtained constant still belongs to the Herbrand universe, the same symbol is added again. This process proceeds as long as we receive a distinct constant.

2. Apply the elementary substitution instance to the positive part of the rule antecedent. The obtained set of facts is an elementary test situation. If we take an example from an SLA, the elementary test situation generated from *maintenance*(*Service*) will be *maintenance*(*service*). It is obtained by replacing the variable *Service* with the constant *service*. Such constant can be understood as a representative of possible values of the variable.

3. If KB_2 contains rules with negative literals that satisfy the condition of the part 2 of the Definition 5, construct additional elementary test situations, using various combinations of facts from negative parts of the rules, according to the part 2 of the Definition 5.

In order to compare knowledge bases (that represent SLAs in our case), we need to construct elementary test situations from all rules, and then generate models from them in different knowledge bases. The result of the comparison is a set of those test situations that lead to different conclusions, and the corresponding difference sets, as defined in Section 3.3.

Example 5. The example below shows two knowledge bases and models, generated from different elementary test situations. The situations were constructed according to the algorithm above, and models are generated using model generating algorithm described in Section 3.2. There are four elementary test situations: $\{S(x)\}$, $\{S(x), P(a)\}$, $\{S(x), R(x)\}$, and $\{S(x), P(a), R(x)\}$. As we can see, the knowledge bases are different (have different conclusions) in the situation $\{S(x), P(a)\}$ only. For this test situation, the conclusion $Q(x)$ is obtained only in KB_1, and the conclusion $R(x)$ is obtained only in KB_2.

KB_1	KB_2
$R(X) \leftarrow S(X)$, not $P(a)$ $Q(X) \leftarrow S(X)$, not $R(X)$	$R(X) \leftarrow S(X)$
$\{S(x)\} \cup KB_1 \vdash_M \{S(x), R(x)\}$ $\{S(x), P(a)\} \cup KB_1 \vdash_M \{S(x), P(a), Q(x)\}$ $\{S(x), R(x)\} \cup KB_1 \vdash_M \{S(x), R(x)\}$ $\{S(x), P(a), R(x)\} \cup KB_1 \vdash_M \{S(x), P(a), R(x)\}$	$\{S(x)\} \cup KB_2 \vdash_M \{S(x), R(x)\}$ $\{S(x), P(a)\} \cup KB_2 \vdash_M \{S(x), P(a), R(x)\}$ $\{S(x), R(x)\} \cup KB_2 \vdash_M \{S(x), R(x)\}$ $\{S(x), P(a), R(x)\} \cup KB_2 \vdash_M \{S(x), P(a), R(x)\}$

3.5. Theorems about Elementary Test Situations

In this section, we present theorems about the sufficiency of using elementary test situations in order to find differences between knowledge bases. The first theorem is formulated for Horn logic programs, and the second one is for non-Horn logic programs.

Theorem 1. *Let KB_1 and KB_2 be knowledge bases defined as Horn logic programs. Then KB_1 and KB_2 are different if and only if there is an internal difference between them or they are different in an elementary test situation.*

Proof. Sufficiency of the condition is obtained directly from Definition 3.

Let's prove necessity of the condition. According to Definition 3, knowledge bases are different in two cases: (a) if there is an internal difference between them, or (b) if they are different in some situation S. The first case is repeated in the formulation of the theorem, so this case is proved. Let's prove the theorem in the second case, using proof by contradiction.

Let $d \in Diff(S, KB_1, KB_2)$, and d is generatable from $S \cup KB_1$ and not generatable from $S \cup KB_2$. Let's construct a sequence $T=[t_0, \ldots, t_n]$ of ground substitution instances of KB_1 clauses such that: (a) for each i ($0 \le i \le n$) and for each antecedent element A_{ij} of the clause t_i, either $A_{ij} \in S$ or A_{ij} is an internal fact of KB_1 or there exists $k<j$ such that A_{ij} is a consequent of t_k (that is the rules are linked in a chain), (b) d is a consequent of t_n.

Let's prove that at least one of the following statements is true: (1) in sequence T, there exists $C_S \leftarrow A_S$ such that A_S is an internal fact of KB_1 and does not belongs to the model of KB_2, (2) in sequence T, there exists $C_S \leftarrow A_S$ such that if $A_S \cup KB_2 \vdash_M M$ then $C_S \notin M$. This is true, because otherwise d would be generatable from $S \cup KB_2$, contrary to our assumption that $d \in Diff(S, KB_1, KB_2)$.

The truth of the first statement means that the knowledge bases have internal difference, and the theorem is proven in this case.

Let's consider the second statement. Let $C_S \leftarrow A_S$ be obtained from $C \leftarrow A$ using a substitution σ. Let's prove, that if we replace this substitution by a substitution θ such that $A\theta$ is an elementary substitution instance of formula A, then $C\theta$ is not generatable from $A\theta \cup KB_2$ just as $C\sigma$ is not generatable from $A\sigma \cup KB_2$. Then the theorem will be proven, and $E = A\theta$.

Let's consider different cases of an element X/t (X – variable, t – term) of the substitution. We use notation $P(\ldots X \ldots)$, where P is a predicate name, and X is occurrence of the variable, for example, $P(X)$, $P(X,Y)$, $P(b, f(a,X), c)$, etc. Let's consider what is the substitution instance of $P(\ldots X \ldots)$ for different types of t in the substitution X/t, and what formulas can be unified with it:

t	$P(...X...)\{X/t\}$	Can be unified with
$x, x \notin U_H$	$P(...x...)$	$P(...Z...)$
$a, a \in U_H$	$P(...a...)$	$P(...Z...)$, $P(...a...)$
$f(x)$	$P(...f(x)...)$	$P(...Z...)$, $P(...f(Z)...)$
$f(a)$	$P(...f(a)...)$	$P(...Z...)$, $P(...f(Z)...)$, $P(...f(a)...)$
$g(f(x))$	$P(...g(f(x))...)$	$P(...Z...)$, $P(...g(Z)...)$, $P(...g(f(Z))...)$
$g(f(a))$	$P(...g(f(a))...)$	$P(...Z...)$, $P(...g(Z)...)$, $P(...g(f(Z))...)$, $P(...g(f(a))...)$
…	…	…

Here U_H is the Herbrand universe of the union of knowledge bases $KB_1 \cup KB_2$. We see that if an element X/t of the substitution (t is an arbitrary term) is changed to X/x (x is a constant that does not belong to Herbrand universe), we will have less formulae that can be unified with the obtained substitution instance. Consequently, number of applicable rules can decrease, but not increase. Additionally, the set of applicable rules can be narrowed if the initial substitution replaces different variable by the same term, whereas we replace them with distinct terms. As an example let's consider $P(X,Y)\{X/t,Y/t\}=P(t,t)$. This substitution instance can be unified with $P(Z,V)$ and $P(Z,Z)$, whereas $P(X,Y)\{X/x,Y/y\}=P(x,y)$ and this substitution instance can be unified only with $P(Z,V)$. So, the set of rules that can be applied to the situation $A\theta$ at the first step of the model generating is a subset of rules which are applicable for the situation $A\sigma$ at the first step. After we apply these rules we receive the same conclusions as in the case of the initial substitution, but we will have X/x instead of X/t. Therefore, the set of later applicable rules can also shrink, but not expand. If we continue, we will see that the generated model is a subset of the initial model, only with another substitution for X. Analogical consideration of other elements of the substitution θ will lead to the conclusion that if $C\sigma$ is not generatable from $A\sigma \cup KB_2$, then $C\theta$ is not generatable from $A\theta \cup KB_2$. The theorem is proven.■

We may conclude that an elementary substitution instance of every rule antecedent is a representative of situations for which model generating is passing through these rules. In this sense, the whole set of such instances covers the set of all possible situations. To the aim of the difference finding, the check of the elementary instances can be used instead of the check of all possible situations.

Theorem 2*. Let KB_1 and KB_2 be knowledge bases defined as non-Horn logic programs and satisfying the restrictions of bound variables, exactly one model and finite model. Then KB_1 and KB_2 are different if and only if there is an internal difference between them or they are different in an elementary test situation.*

Outline of proof. Sufficiency of the condition is obtained directly from Definition 3. If there is an internal difference between the knowledge bases, necessity of the condition is also obtained directly from Definition 3. If there is no internal difference, the necessity can be proved by contradiction. Let's assume that the knowledge bases are not different in any elementary test situation, but different in some other situation. This difference can be caused by the case when negative parts of one or more rules have facts that prevent the consequent from being included in the model. However, these negative parts are included in corresponding elementary test situations; therefore, there would also be a difference in these situations, contrary to our assumption. Sufficiency of using elementary substitution instances in test situations can be proven analogically as in Theorem 1, i.e. by analyzing conclusions from different types of substitutions. ■

4. Conclusions

The proposed method of the comparison of SLAs makes use of elementary test situations for detecting differences between SLAs. In order to get all conclusions derivable from a particular situation and to compare them, we use formal proof that is based on model generating. We concluded that an elementary substitution instance of every rule antecedent is a representative of situations for which the model generating is passing over these rules, and the whole set of such instances covers the set of all possible situations. To the aim of the

difference finding, the check of the elementary instances can be used instead of the check of all possible situations.

The detection of differences is useful not only for the choosing of a SLA, but also for the negotiating of SLAs and managing changes in SLAs. Indeed, one change can cause outcomes that are not observable at once, because they are derivable by rule chains.

Such comparison is not expedient as a frequently executed operation. Model generating can cause problems in real-time use because lots of rules fire in any cycle and many conclusions are drawn, leading to a lot of redundancy in the memory-based fact base. However, the method can be used before establishing long-term SLAs. Results of the comparison are presented for the analysis that can be performed by a human or by an intelligent agent according some stated criteria.

5. Acknowledgement

This work has been supported by the project "Theoretical and engineering aspects of e-service technology development and application in high-performance computing platforms" (No. VP1-3.1-ŠMM-08-K-01-010) funded by the European Social Fund.

References

1. Alkandari, F., Paige, R.F.: Modelling and Comparing Cloud Computing Service Level Agreements. In: Proceedings of the 1st International Workshop on Model-Driven Engineering for High Performance and Cloud Computing (MDHPCL'12). pp. 3:1–3:6. ACM New York (2012)
2. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M.: Web Service Agreement Specification (WS-Agreement). Open Grid Forum (2007)
3. CSMIC (2014), http://csmic.org. Accessed July 08, 2014
4. Erl, T.: Service-Oriented Architecture (SOA): Concepts, Technology, and Design. Prentice Hall (2005)
5. Garg, S.K., Verseteg, S., Buyya, R.: SMICloud: A Framework for Comparing and Ranking Cloud Services. In: Proceedings of the 4th IEEE International Conference on Utility and Cloud Computing (UCC 2011). pp. 210–218. IEEE Computer Society (2011)
6. Haq, I.U., Paschke, A., Schikuta, E., Boley, H.: Rule-Based Workflow Validation of Hierarchical Service Level Agreements. In: 2009 Workshops at the Grid and Pervasive Computing Conference. pp. 96–103. IEEE (2009)
7. Kyriazis, D. (editor): Cloud Computing Service Level Agreements - Exploitation of Research Results. European Commission, Directorate General for Communications Networks, Content and Technology (2013)
8. Ludwig, H., Keller, A., Dan, A., King, R.P., Franck, R.: Web Service Level Agreement (WSLA) Language Specification. IBM Corporation (2003)
9. Maurer, M., Emeakaroha, V.C., Brandic, I., Altmann, J.: Cost–benefit analysis of an SLA mapping approach for defining standardized Cloud computing goods. Futur. Gener. Comput. Syst. 28 (1), 39–47 (2012)
10. Moghaddam, M., Davis, J.G.: Service Selection in Web Service Composition: A Comparative Review of Existing Approaches. In: Web Services Foundations'. pp. 321–346. Springer-Verlag (2014)
11. Paschke, A.: RBSLA: Rule-Based Service Level Agreements – knowledge representation for automated E-contract, SLA and policy management. Institut für Informatik der Technischen Universität München (2007)
12. Paschke, A., Bichler, M.: Knowledge representation concepts for automated SLA management. Decis. Support Syst. 46 (1), 187–205 (2008)