On Automated Generation of Web Service Level Agreements

Cinzia Cappiello, Marco Comuzzi, and Pierluigi Plebani

Dipartimento di Elettronica e Informazione – Politecnico di Milano
Piazza Leonardo da Vinci 32, 20133 Milano (Italy)
{cappiello,comuzzi,plebani}@elet.polimi.it

Abstract. Before a service invocation takes place, an agreement between the service provider and the service user might be required. Such an agreement is the result of a negotiation process between the two parties and defines how the service invocation has to occur. Considering the Service Oriented Computing paradigm, the relationship among providers and users is extremely loose. Traditional agreements are likely to concern long term relationships and to be manually performed. In this paper, we propose a model to generate service level agreement on-the-fly. Just before the invocation commences, the quality of the service is negotiated in order to generate a service level agreement tied to that specific invocation. Such an approach relies on a quality model that supports both users requirements and providers capabilities definition.

1 Introduction

Organizations are increasingly exporting their services as Web services [1]. Such a proliferation increases the likelihood that users may find several services satisfying their functional requirements [2–4]. When users can choose among a set of functionally equivalent services, non-functional requirements become the driver for Web service selection. As a consequence, we need to define and manage Service Level Agreements (SLAs) between service providers and users [5].

In Service Oriented Computing paradigm, an SLA is defined as a binding contract which formally specifies user expectations about the solution and tolerances. SLA is a collection of service level requirements that have been negotiated and mutually agreed upon by the information providers and the information consumers. Usually, providers define some service levels as a fixed combination of their specific capabilities on a set of quality dimensions, and users must choose one these levels. Reasonable service levels that meet user requirements can be achieved by increasing the flexibility of the SLA definition. We argue that this could be obtained by allowing parties, i.e., users and providers, to re-examine and to negotiate defined levels. It is worth noting that identifying attainable service levels is a time consuming activity for the providers. Adding negotiation features creates further overhead during SLA definition activity. For these reasons, our approach does not identify service levels in advance. Providers only clarify their capabilities and service levels will be identified on-the-fly considering the users

expectations. Service levels negotiation is also performed on-the-fly to reduce its overhead.

The discussion of mechanisms for on-the-fly generation of the SLA will be tied to a running example. We focus on a *TrafficMonitoring* service example. The *TrafficMonitoring* Web service provides up-to-date information about local traffic to business and retail customers across the US. The quality of such a service is defined by two classes of quality dimensions: *technical* and *domain dependent*.

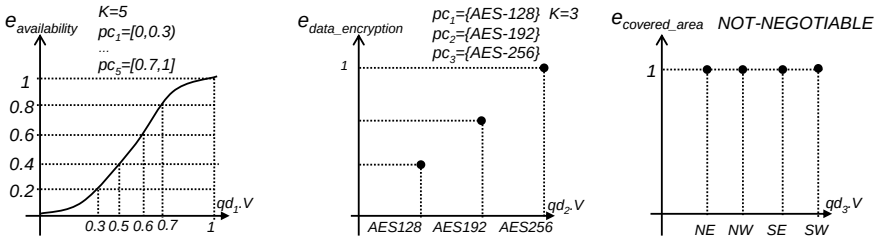
Technical quality dimensions refer to technical aspects of service provisioning. Quality dimensions belonging to this class can be associated with any Web service, and do not explicitly depend on a characterization of the domain in which a Web service operates. For the sake of simplicity, we consider three quality dimensions, that is, *availability*, *data encryption*, and *response time*. Readers may refer to [6, 7] for an extensive review of Web service technical quality. Availability refers to the expected percentage of time the system is up and accessible. Data encryption refers to the algorithms adopted for protecting data from malicious accesses. Eventually, response time refers to the expected delay between the moment in which a request is sent and the moment in which results are received [6].

Domain dependent quality dimensions strongly rely on the type of Web service that is under consideration. For the *TrafficMonitoring* example, we consider the *covered area*, *routes set*, and *detail level* dimensions. The *covered area* dimension characterizes the extensiveness of the area over which the service is able to provide traffic information. A service, for instance, may provide information only on national highways, while other ones may also consider interstate or local routes and downtown traffic conditions. Similarly, the detail level of traffic information provided by a service may also vary. A service may provide information on accidents and traffic jams, while other ones may also provide information about closed routes, detours, and predictions about future conditions of local traffic.

The paper is organized as follows. Section 2 presents a model to describe Web service quality, provider capabilities, and user requirements. Section 3 describes the negotiation model by which SLAs can be obtained on-the-fly. Section 4 discusses related work, while conclusions are finally drawn in Section 5.

2 Quality model

A negotiation process occurs whenever both a user and a provider are able to define the documents specifying the requirements and the capabilities, namely. In a Web service environment, where users and providers might not know each others in advance, these documents must rely on the same language. In [8], a model able to express the quality of a Web service is discussed. The same model, discusses in the following, will be adopted in this work as well.



(a) *Availability* (b) *Data Encryption* (c) *Covered area*

Fig. 1. Evaluation functions and primitive service classes for *availability*, *data encryption*, and *covered area*.

The quality of a Web service is defined by a set of quality dimensions [1] each of them associated to a given quality aspect. More formally, we define a quality dimension qd_i as:

$$qd_i = \langle name, V, ef(V), PC \rangle \;\; i = 1, \ldots, I. \tag{1}$$

The *name* uniquely identifies the quality dimension. The element V corresponds to either categorical or interval admissible values. In the former case, the admissible values will be included in a specific vector $V = \{v_h\}$ $(h = 1, \ldots, H)$, while, in the latter case V will be defined by its extremes, i.e., $V = [v_{min}, v_{max}]$. The function $ef : V \rightarrow [0..1]$ represents the *quality evaluation function*, i.e., how the quality increases or decreases with respect to the admissible values: 0 means lowest quality, 1 highest quality. The trend of ef is usually defined by an utility function, e.g., linear, logarithmic, exponential, sigmoidal. The admissible value set V is organized in disjoint primitive service classes $PC = \{pc_k\}$ $(k = 1, \ldots, K)$ and are obtained as follows:

- In case of categorical values, the primitive service classes coincide with the values that the dimension may assume: i.e, $qd_i.PC \equiv qd_i.V \;\; , H = K$.
- In case of interval values, primitive service classes are obtained by splitting $V = [v_{min}, v_{max}]$ into K intervals, so $PC = \{pc_k = [pc_{k_{min}}; pc_{k_{max}}]\}$ where $pc_{k_{max}} = pc_{(k+1)_{min}}$, $pc_{1_{min}} = v_{min}$, $pc_{K_{max}} = v_{max}$. pc_k ranges are obtained as follows: let divide $qd_i.ef(V)$ in K ranges $\{[e_{k_{min}}; e_{k_{max}}]\}$, then $p_{k_{min}} = qd_i.ef^{-1}(e_{k_{min}})$ and $pd_{k_{max}} = qd_i.ef^{-1}(e_{k_{max}})$.

Figure 1(a) and 1(b) show, respectively, this methodology applied to the *availability* and *data encryption* dimensions in the running example. The definition of primitive service classes is exploited by the negotiation algorithms described in Section 3. We assume that additional elements, such as measurement units or metrics, are also defined. We do not explicitly include them in qd_i since they are not relevant for our approach

[1] In the literature, quality dimensions are also named quality attributes or quality parameters.

Given a Web service, its quality is defined by the set $QD = \{qd_i\}$. As mentioned above, negotiation takes place only if both requirements and capabilities are expressed on the same quality dimensions set. For this reason we assume that a third party, called *community*, is in charge of identifying the set of relevant quality dimensions. In this way, the quality dimensions included in QD will be used (i) by the provider to express the offered quality, i.e., *capabilities* C and (ii) by the user to define the required quality, i.e., *user requirements* UR.

As defined in [9], a community is a group of people which aims at proposing a specification for a group of objects with some relevant common characteristics. More generally, given an application domain, we suppose that a community exists and produces the set of relevant quality dimensions. Sometimes, the community can be easily identified since it is explicitly constituted (e.g., tourism community, financial community). Most of the times the community associated with an application domain does not explicitly exist. For example, if we want to buy a laptop then everyone can list the set of relevant quality dimensions which the evaluation of the laptop quality relies on, e.g., CPU, memory, HD capacity, screen resolution, and so on. Roughly speaking, the agreement on QD between providers and users definitely exists but it is implicit. In some way, introducing the actor community means to make explicit this implicit common understatement. s

Table 1 shows the quality dimensions included in QD for the *TrafficMonitoring* example. Once the community decides to include a qd_i in QD, the community also defines the range of admissible values, the related evaluation function $qd_i.ef$, and the primitive service classes $qd_i.PC$. In Table 1, all the $qd_i \in QD$ are described. In some case (e.g., covered area), the community cannot state which are the best and worst values, since they depend on the user preferences. So, the evaluation function always returns 1. This kind of dimensions, as explained in Section 3, are non-negotiable.

Table 1. Quality parameters for Traffic Monitoring example.

name	**V**	**ef**	**P**
availability	[0,1]	sigmoidal (see Figure 1(a))	$\{[0, 0.3); [0.3, 0.5); \ldots; [0.7, 1]\}$
data encryption	[AES-128;AES-192;...]	linear	[AES-128;AES-192;...]
response time	[0,10]	inverse linear	$\{[0.2, 1], \ldots, [9, 10]\}$
covered area	[SouthEast;SouthWest; NorthEast;NorthWest]	1 $\forall v_h \in V$ (see Figure 1(c))	[SouthEast;...]
routes set	[Highways;interstate; local;...]	1 $\forall v_h \in V$	[Highways;interstate; local;...]
detail level	[jams; detours; toll;...]	1 $\forall v_h \in V$	[jams; detours; toll;...]

It is worth noting that the range of admissible values has been identified regardless of a specific Web service implementation. So, we assume that all the

existing Web services, given a quality dimension, can only offer a subset of the admissible values defined by the community. In addition, users will customize the quality dimensions accordingly to their preferences.

Starting from the QD defined by the community, Sections 2.1 and 2.2 describe, respectively, how the capabilities and the requirements can be defined.

2.1 Capabilities

Capabilities reflect the quality offered by a Web service provider. Focusing on the service description, the provider before publishing its Web service will define a document expressing the functional aspects. About this, WSDL represents the *de-facto* standard that identifies the set of available operations and exchanged messages. Along with the functional aspects, the service provider also needs to attach a document in which the offered quality is described. At this stage, the literature does not include a language for quality description with the same consensus as WSDL does for the functional aspects. Anyway, we think that the capabilities as introduced in the following can be simply expressed according to languages such as WSOL [10] or WS-Policy [11].

We define a capability $c(qd_i)$ as a restriction on the range of admissible values of the quality dimension qd_i. More precisely:

$$c(qd_i) = \langle qd_i.name, offering, qdprice(offering) \rangle, \tag{2}$$

where $offering \subseteq qd_i.V$ represents the restriction on the range of admissible values. In this way, the provider defines, given a quality dimension, which are the actual values the provider is able to support. In addition, the provider also defines *qdprice* function which maps the dependency between the offered values and the price per user associated with such a provisioning.

According to this model, the provider during the publication process of a Web service, will attach a document C collecting all the supported capabilities. In particular:

$$C = \{c(qd_i)\} \quad \forall qd_i \in QD. \tag{3}$$

In other words, a capability document must include all the quality dimensions previously identified by the community. Table 2 lists the capabilities of a hypothetical *TrafficMonitoring* service provider. For instance, the offered *availability* is included in the range $[0.5, 1.0]$ and the price for such a provisioning is given by a fixed amount (e.g., 30$) and a variable one that varies according to the actual value of the availability (e.g., availability*5$). Similarly, different prices will be associated to different *covered area*. Since US NorthEast is more populated than US NorthWest then the price varies accordingly (e.g., 5$ rather than 3$).

2.2 Requirement model

Similarly to the capabilities, the user requirements are expressed on the basis of the quality dimensions identified by the community. In particular, for each

Table 2. Capabilities for TrafficMonitoring service.

qd	**offering**	**qdprice**
availability	[0.5,1.0]	30$+(availability*5$)
data_encryption	[AES-128]	500$
response time	[1,2]	3$*(5$/timeliness)
covered_area	[NorthEast;NorthWest]	5$-NE;3$-NW
route_set	[interstate;local]	5$-interstate;10$-local
detail_level	[detours]	10$

$qd_i \in QD$ users operate a restriction on the admissible range of values. With this operation, the users state which is the required quality. Hence, a user requirement R that will be compared to the capabilities C during to the negotiation process is defined as:

$$UR = \langle \{ur(qd_i)\}, budget\rangle, \tag{4}$$

where the $\{ur(qd_i)\}$ represents the user requirements of a specific qd_i and $budget$ is the amount of money that the user is willing to pay for the service. In detail:

$$ur(qd_i) = \langle qd_i.name, request, w\rangle. \tag{5}$$

Here $request \in qd_i.V$ represents the restriction on the range of admissible values. This restriction corresponds to the values required by the user for the given quality dimension.

The element w in ur represents the weight that identifies how much the related quality dimension qd_i influences the overall quality of the service. It is worth noting that the weight assignment activity is a crucial point of the method. It can be performed in different ways. The simplest way could be to let users associate with each quality dimension a weight to express the importance that the dimension has for the specific user class. In this case the only constraint is that the sum of the weights associated with all the dimensions has to be equal to 1. This method is difficult to apply, since the absolute relevance of a dimension on the total quality is hardly identifiable. For this reason, in this model we assume that the weight assignment is driven by the AHP (Analytic Hierarchy Process) approach, a decision making technique developed by T.L. Saaty [12]. This is a qualitative approach in which the user only states if a sub-dimension is more influent than another one on the overall quality. We assume that all the quality dimensions are independent. AHP is a decision-making technique that assigns to each sub-dimension a score that represents the overall performance with respect to the different parameters. AHP is suitable for hierarchical structures as the quality model described previously and proposes to user pairwise comparisons between sub-dimensions.

Considering the difficulty that some users have in the requirements specification, we assume that the community supports them by preliminarily identifying

their profile. We borrow the *profiling* concept from the Web Information Systems (WIS) literature in which it is used for the personalization of content to user expectations. Profiling is the technique through which data are collected and manipulated with the goal of identifying and describing the profile of an entity, such as a user, an object, a product, or a process [13]. A *profile* is a source of user requirements, in fact it is a structured representation of the information that describes users and their preferences along the services that they require. This information can be obtained by suitable architectures and modules operating along with the Web service infrastructure.

In the requirement model proposed in this paper, users are characterized by a profile and assigned to users classes. Each class contains users with similar characteristics. Formally, our model considers a set $U = \{u_y\}$ of user and a set $UC = \{uc_z\}$ of users classes. In particular, we assume that:

$$\forall u_y \in U \ \exists! uc_z \in UC \mid u_y \in uc_z. \tag{6}$$

A class of user uc_z corresponds to the requirements suitable for the users belonging to the class. According to our model:

$$uc_z = \{ur_z(qd_i)\}. \tag{7}$$

We assume that the community is in charge of defining such requirements, therefore, of identifying the users classes. In this way, users of a class UC have a sort of template of requirements that can be customized with respect to specific requirements to produce a specific UR.

Given a class of users, the user class requirements ur_z represents the quality of service usually required by the user belonging to that given class. Users take inspiration from these requirements defined by the community to express their specific user requirements. User requirements can be more or less selective than class requirements. Table 3 shows possible user requirements given by a user for the *TrafficMonitoring* service. For example, if class requirements for the availability dimensions are the values included in the range $[0.5, 1.0]$, the user can be more selective by specifying a quality limit greater than 0.7 or, alternatively, decrease the relevance of the data quality dimension by accepting a range such as $[0.7, 0.99]$.

Table 3. User requirements for TrafficMonitoring service.

qd	**request**	**w**
availability	[0.5,1.0]	0.4
data_encryption	[AES-128]	0.025
response time	[0.5,1]	0.3
covered_area	[SouthEast;NorthEast]	0.1
route_set	[highways;local]	0.15
detail_level	[jams;detours]	0.025

3 Negotiation model

Before negotiation taking place, we need to state if the offerings satisfy the user requirements. So, we have to verify the following statement:

$$\forall qd_i \in QD \;\; isec\,(c(qd_i), ur(qd_i)) = c(qd_i).offerings \cap ur(qd_i).request \neq \emptyset. \quad (8)$$

The service level negotiation occurs within the quality values identified by $isec\,(c(qd_i), ur(qd_i))$.

Automated negotiation is usually defined by three elements: the *negotiation protocol*, the participants *decision models* [14], and the *negotiation objects*. We adopt a very simple negotiation protocol where, the user for each qd_i starts considering the primitive class in $qd_i.PC$ which also belongs to the calculated intersection and which corresponds to the lowest quality. Then, as long as the budget is not fully exploited, the user will consider the primitive class with higher quality. In this mechanism, the decision model controls the way in which the budget is split across the quality dimensions. Finally, negotiation objects refer to the elements over which negotiation is performed. We argue that only the QoS dimension associated to a non-constant evaluation function $qd_i.ef$ are negotiable. For dimensions characterized by a constant evaluation function, e.g., *covered area* in our running example, we hypothesize that the user's requirements are *non-negotiable*. If, for instance, the user identifies NE and NW as required values for the *covered area* dimensions, the user requests can be fulfilled only when the service provides traffic information on NE and NW. We hypothesize that it is not possible to negotiate on this kind of dimensions, since the community is not able to define an evaluation function that orders their values. Therefore, the set QD is split in two sets: the set NQD of negotiable quality dimensions, and the set $NNQD$ of non-negotiable quality dimension. More formally:

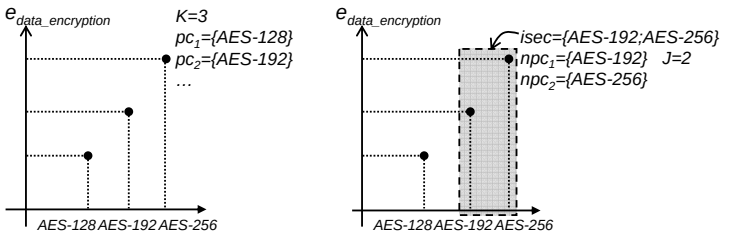
$$QD = NQD \cup NNQD, \;\; NQD \cap NNQD = \emptyset$$
$$NQD = nqd_l \quad l = 1, \ldots, L$$
$$NNQD = nnqd_m \quad m = 1, \ldots, M$$

For each negotiable quality dimension nqd_l, we formally define the negotiation objects as:

$$negobj_l\,(c(nqd_l), ur(nqd_l)) = \langle nqd_l.name, NPC, ur(nqd_l).w \rangle \quad l = 1, \ldots, L. \quad (9)$$

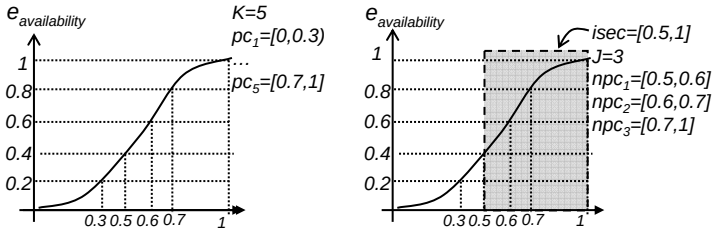
As mentioned above, for each quality dimensions we calculate the intersection of related capabilities and user requirements. Since qd_i is divided by definition into K primitive classes, then only a subset of them will be included in the intersection as well. Such a subset is named NPC (negotiation primitive classes) and defines, for each quality dimension nqd_l, the set of negotiation service classes nsc_j, $j = 1, \ldots, J$ included in the intersection ($J \leq K$). The set NPC includes also the price $price(npc_j)$ associated by the service provider to each negotiation service class:

$$NPC(nqd_l) = \{\langle npc_j, price(npc_j) \rangle\} \;\; j = 1, \ldots, J \;\; l = 1, \ldots, L. \quad (10)$$



(a) Primitive service classes (b) Negotiation service classes

Fig. 2. Defining negotiation service classes for *data encryption*.



(a) Primitive service classes (b) Service classes for negotiation

Fig. 3. Defining negotiation service classes for *availability*.

The methodology for defining service classes npc_j and their price differs with respect to the nature of the negotiable quality dimension nqd_l.

As reported in Section 2, when considering a dimension nqd_l that assumes categorical values, the primitive service classes $nqd_l.PC$ coincide with the values $nqd_l.V$ identified by the community. Figure 2 shows the methodology to obtain negotiation service classes for the *data encryption* dimension. In this case, the price associated with a service class npc_j is directly obtained from the price information in the provider capabilities. A negotiation service class npc_j includes, in fact, one single value $v_{\bar{h}} \in V$, hence:

$$price(npc_j) = c(nqd_l).qdprice(nqd_l.v_{\bar{h}}). \tag{11}$$

The definition of negotiation service classes npc_j for continuous nqd_l derives from the restriction operated on primitive service classes $nqd_l.PC$ over $isec\,(c(nqd_l), ur(nqd_l))$. How to obtain service classes for the *availability* dimension is graphically reported in Figure 3. Let us refer to $min(npc_j)$ and $max(npc_j)$ as, respectively, the left and right boundaries of the negotiation service class npc_j. The price $price(npc_j)$ associated with a service class npc_j is the average between the price associated with its left and right boundaries, that is:

$$price(npc_j) = \frac{c(nqd_l).qdprice[min(npc_j)]) + c(nqd_l).qdprice[max(npc_j)])}{2}. \tag{12}$$

The algorithm adopted to assign a price to a service class can be more general and it is usually defined by the community.

Once having defined $negobj_l\,(c(nqd_l), ur(nqd_l))$, $\forall l,\ l = 1, \ldots, L$, we define the basic quality level QL_{base} of the Web service, which is constituted, for each negotiable quality dimension, by the lowest quality negotiation service class $negobj_l.npc_1$. Then, it will be:

$$QL_{base} = \{negobj_1.npc_1, \ldots, negobj_L.npc_1\}\,. \tag{13}$$

The objective of the negotiation is to obtain a negotiated quality level QL_{neg} which improves the quality of the basic level. The user exploits the declared budget $ur(nqd_l).budget$ to configure the basic quality level and increase the expected quality of the Web service. The price $P(QL_{base})$ associated to the basic quality level is:

$$P(QL_{base}) = \sum_{l=1}^{L} price(npc_1(nqd_l)). \tag{14}$$

Let us define P_{nn} as the price associated with the quality values assumed by non-negotiable dimensions in $isec\,(c(nnqd_m), ur(nnqd_m))$. In the running example, the community may assume that *covered area*, *routes set*, and *detail level* are non-negotiable ($M = 3$). Let us consider a user requirement that specifies *highways* and *local* as required values for the *routes set* dimension. A SLA between a service provider and the user can be generated only if the provided Web service gives traffic information on highways and local routes. If we assume that the user has also required *jams* and *NE* for, respectively, *detail level* and *covered area*, it will be:

$$\begin{aligned} P_{nn} = {} & c(nnqd_1).qdprice_{covered\_area}(NE) + \\ & + c(nnqd_2).qdprice_{detail\_level}(jams) + \\ & + c(nnqd_3).qdprice_{routes\_set}(highways) + \\ & + c(nnqd_3).qdprice_{routes\_set}(local). \end{aligned} \tag{15}$$

We can now define the extra budget EB of the user as:

$$EB = budget - [P(QL_{base}) + P_{nn}]\,. \tag{16}$$

If $EB < 0$, then the service is not going to be provisioned because the user is not able to cover with the budget the total price of the service, that is, the sum of the price associated with the basic quality level for negotiable dimensions and the price of non-negotiable dimensions. In case $EB = 0$, then the service will be provisioned with the basic quality level QL_{base} for negotiable quality dimensions. The negotiation does not take place. The negotiation is executed only if $EB > 0$. Two strategies are available to the user to decide how to split EB across the different negotiable quality dimensions, that we name the *vertical* and the *horizontal* strategies.

```
01  define ΔEB_l = 0, ∀l //Fraction of EB allocated to the
                          //improvement of nqd_l
02  define ΔEB = 0 //Exploited fraction of the extra budget
03  while(END==FALSE)
04    select l:max(nqd_l.w) = w_l //select the current nqd with
                                  //highest priority
05    w_l = w_l − 0.01 //decrease the priority of the selected nqd
06    ΔEB_l = price(npc_{j+1}) − price(npc_j) //update EB allocation
                                              //on nqd_l
07    npc_j(nqd_l) = npc_{j+1}(nqd_l) //update the nqd_l level
08    ΔEB = ΔEB + ΔEB_l //update the EB allocation
09    if (ΔEB > EB) //Cannot price increase be covered by EB?
10       npc_j(nqd_l) = npc_{j−1}(nqd_l) //restore old nqd_l value
11       ΔEB = ΔEB − ΔEB_l //restore EB allocation
12       END=TRUE //Exit condition, negotiation stops
13    endif
14    if (w_l == 0) //Exit condition, negotiation stops
15       END=TRUE
16  endwhile
```

Fig. 4. Horizontal negotiation strategy.

When adopting the vertical strategy, the user has the objective to maximize the quality associated to the highest priority dimension $nqd_{\bar{l}}$. When the quality of this dimension is maximized, that is, when the remaining extra budget exceeds the price of the negotiation service class $npc_J(nqd_{\bar{l}})$, then the algorithm switches to the maximization of the quality of the second highest priority dimension. The horizontal strategy is adopted when the user wants to split the extra budget on the negotiable quality dimensions proportionally to the priorities $ur(nqd_l).w$, $\forall l \in [1, \ldots, L]$. The horizontal and vertical strategies follow respectively, the algorithms reported in Figure 4 and 5.

Let us refer to P as the total price of a service after quality negotiation:

$$P = P(QL_{neg}) + P_{nn}. \tag{17}$$

The result of the negotiation is a service level agreement SLA, generated on-the-fly for Web service, that has the following structure:

$$SLA = \langle QL_{neg}, P, isec\left(c(nnqd_m).ur(nnqd_m)\right)\rangle, \tag{18}$$

where QL_{neg} reports the service class for negotiable quality dimensions obtained from the execution of negotiation, P is the total price associated with the Web service with negotiated quality. Last term refers to the values of the non negotiable quality dimensions.

```
01  define ΔEB_l = 0, ∀l //Fraction of EB allocated to the
                          //improvement of qd_i
02  define ΔEB = 0 //Exploited fraction of the extra budget
03  while(END==FALSE)
04    select l:max(nqd_l.w) = w_l //select the current nqd with highest
                                  //priority
05    w_l = w_l − 0.01 //decrease the priority of the selected nqd
06    STOP=FALSE //starting configuration of nqd_l
07    while (STOP==FALSE)
08       ΔEB_l = price(npc_{j+1}) − price(npc_j) //update EB allocation on
                                                 //nqd_l
09       npc_j(nqd_l) = npc_{j+1}(nqd_l) //update the nqd_l value
10       ΔEB = ΔEB + ΔEB_l //update the EB allocation
11       if (ΔEB > EB) //Cannot price increase be covered by EB?
12          npc_j(nqd_l) = npc_{j−1}(nqd_l) //restore old nqd_l value
13          ΔEB = ΔEB − ΔEB_l //restore EB allocation
14          STOP=TRUE //end nqd_l negotiation
15          END=TRUE //exit condition, negotiation stops
16       endif
17       if((j = J)OR(w_l == 0))
18          STOP=TRUE //end nqd_l configuration
18    endwhile
19  endwhile
```

Fig. 5. Vertical negotiation strategy.

4 Related work

This paper presents a model to support the automatic generation of a service level agreement by considering user requirements and provider capabilities. To mediate between these two standpoints, we introduce the community as the actor able to provide a shared knowledge about the quality of a service in a specific application domain. The community defines which relevant aspects of a service can be used as search discriminants in service discovery. In the paper, the community organizes dimensions by using a tree-based structure. This approach for defining service quality has been inspired by [15] and [16], which recognize the correlation among several dimensions. In particular, [15] also refers dimensions to different layers (i.e. system level, resource level, and application level).

The set of dimensions identified by the community is also used as a guideline by the providers to describe the capabilities of the offered service. In fact, a complete service description is an important requirement for users who aim at searching the most suitable Web service. Besides the functional description, for which WSDL represents the most adopted specification, non functional specifications have to be modeled. In [17] a complete comparison of the current quality

description languages is presented. Among all the identified contributions, for our work it is important to consider proposed languages for offers and contracts and languages for policies. As regards the former category, WSOL [10], WSLA [18], and WS-Agreement [19] provide some description models that our work can exploit to express quality dimensions. These contributions are particularly relevant, since they also address the definition and monitoring of quality levels. WSOL is suitable for the definition of quality dimensions, their metrics and quality constraints. The language does not formalize the contract terms between user and provider defining service levels but it contains constructs to define simple quality constraints on each quality dimension. A support for the definition and monitoring of Service Level Agreements is, instead, provided by the WSLA language. It allows providers to define quality dimensions and to describe evaluation functions. Furthermore, it provides monitoring of the parameters during operations and invocation of recovery actions when contract violations occur. Similarly, WS-Agreement provides constructs for advertising the capabilities of providers and for creating agreements based on creational offers, and for monitoring agreement compliance at runtime. The latter category includes WS-Policy [11] that can be adopted as a language for defining capabilities and requirements. WS-Policy definitions are independent of any specific quality descriptions. Using this language, users may describe services by using self-defined quality attributes.

Once that the service capabilities description is provided, the selection of the most suitable service is enabled by the definition of the user requirements. In this area, notations and languages to express users requirements have been defined in NoFun [20] and QML [21]. There are also contributions in which quality requirements are expressed by means of standard sentences or linguistic patterns in natural language [22].

In this paper, the automatic generation of a service level agreement is enabled by the use of negotiation mechanisms. In the literature, the only examples that propose policies for automated quality negotiation of Web services can be found in [23, 24]. In general, research on SLA management has been carried out in the past couple of years and it has been mainly focused on the SLA specification and on the definition of languages for SLA creation, operation, monitoring, and termination. Examples of SLA management frameworks are WS-agreement [19], WS-negotiation [25], and the Service Negotiation and Acquisition Protocol (SNAP) [26]. However, while these standards are still evolving, they present some limitations. Generally, frameworks for SLA management only define the format and types of messages that can be used in the negotiation, but they do not provide the strategies through which negotiation is performed. In this paper, besides a characterization of negotiation messages built on the underlying Web service quality model, we also define the users' strategies to be adopted in the negotiation.

5 Conclusions and future work

This paper proposed a framework for the on-the-fly generation of Web service SLAs. The contribution of the paper is twofold. First, we introduced a quality model for Web services that is exploited by providers and users to define, respectively, their capabilities and requirements. Secondly, we provided users with a mechanism to negotiate among the set of service classes at the intersection between capabilities and requirements.

From the quality model definition perspective, future work should deal with an extended multi-level hierarchical model that considers composite dimensions, such as, for instance, security defined as a combination of data encryption, authentication, non-repudiation, and data integrity. Concerning negotiation, this paper focused on SLA generation involving only one provider and one user. Future work should also investigate how negotiation of quality aspects can be used to select a service among a set of functionally equivalent services. In this way, we will be able to add on-the-fly SLA generation capabilities to the common frameworks dealing with service discovery.

Acknowledgment

The work has been partially supported by the Italian MIUR-FIRB TEKNE Project and by the European WS-DIAMOND Project.

References

1. Papazoglou, M.P., Georgakopolous, G.: Service Oriented Computing: Introduction. Communications of the ACM **46**(10) (2003) 1–5
2. Bianchini, D., De Antonellis, V., Pernici, B., Plebani, P.: Ontology-based methodology for e-service discovery. Information Systems **31**(4-5) (2006) 361–380
3. Bernstein, A., Klein, M.: Towards High-Precision service retrieval. In: Proc. Int. Semantic Web Conference, ISWC'02. (2002)
4. Stroulia, E., Wang, Y.: Structural and semantic matching for assessing web-service similarity. Int. J. Cooperative Inf. Syst. **14**(4) (2005) 407–438
5. Keller, A., Ludwig, H.: The WSLA framework: Specifying and monitoring service level agreements for Web services. Journal of Network and Systems Management **11**(1) (2003) 57–81
6. Ran, S.: A model for Web services discovery with QoS. ACM SIGCOM Exchange **4**(1) (2003) 1–10
7. Mani, A., Nagarajan, A.: Understanding quality of service for Web services. Technical report, IBM, http://www-128.ibm.com/developerworks/library/ws-quality.html (2002)
8. Fugini, M., Plebani, P., Ramoni, F.: A user driven policy selection model. In: ICSOC '06: Proceedings of the 4th international conference on Service oriented computing. To appear. (2006)
9. Marchetti, C., Pernici, B., Plebani, P.: A quality model for multichannel adaptive information. In: WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters, New York, NY, USA, ACM Press (2004) 48–54

10. Tosic, V., Ma, W., Pagurek, B., Esfandiari, B.: Web Service Offerings Infrastructure (WSOI) - a management infrastructure for XML Web services. In: Network Operations and Management Symposium, 2004. NOMS 2004. IEEE/IFIP. Volume 1. (2004) 817–830
11. Vedamuthu, A., Orchard, D., Hondo, M., Boubez, T., Yendluri, P.: Web Services Policy 1.5 - Primer. `http://www.w3.org/TR/2006/WD-ws-policy-primer-20061018` (2006)
12. Saaty, T.L.: The Analytic Hierarchy Process. Mc Graw Hill, New York (1980)
13. Olson, J.: Data Quality: The Accuracy Dimension. Morgan Kaufmann, San Francisco (2002)
14. Jennings, N., Faratin, P., Lomuscio, A., Parsons, S., Wooldridge, M., Sierra, C.: Automated negotiation: Prospects, methods and challenges. Group Decision and Negotiation **10**(2) (2001) 199–215
15. Sabata, B., Chatterjee, S., Davis, M., Sydir, J., Lawrence, T.: Taxonomy for QoS Specifications. In: Object-Oriented Real-Time Dependable Systems, 1997. Proceedings., Third International Workshop on. (1997) 100–107
16. Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic (2000)
17. Ruckert, J., Paech, B.: Web Service Quality Descriptions for Web Service consumers. In: CONQUEST2006. Proceedings. (2006)
18. Keller, A., Ludwig, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. Technical Report RC22456(W0205-171), IBM Research Division, T.J. Watson Research Center (2002)
19. GRAAP Working Group: WS-Agreement Framework. `https://forge.gridforum.org/projects/graap-wg` (2003)
20. Franch, X.: Systematic formulation of non-functional characteristics of software. In: 3rd International Conference on Requirements Engineering (ICRE '98). (1998) 174–181
21. Frølund, S., Koistinen, J.: Quality-of-service specification in distributed object systems. Distributed Systems Engineering Journal **5**(4) (1998)
22. Duran, A., Bernardez, B., Toro, M., Corchuelo, E., Ruiz, A., Perez, J.: Expressing customer requirements using natural language requirements templates and patterns. In: Proceedings of the third Conference on Circuits, Systems, Communications and Computers (CSCC '99). (1999)
23. Lamparter, S., Agarwal, S.: Specification of policies for Web service negotiations. In: Proc. Semantic Web and Policy Workshop. (2005)
24. Gimpel, H., Ludwig, H., Dan, A., Kearney, R.: PANDA: Specifying policies for automated negotiations of service contracts. In: Proc. 1st Int. Conf. Service Oriented Computing, ICSOC'03. (2003) 287–302
25. Rahwan, I., Kowalczyk, R., Pham, H.H.: Intelligent agents for automated one-to-many e-commerce negotiation. In: Computer Science 2002, Twenty-Fifth Australasian Computer Science Conference (ACSC2002). (2002) 197–203
26. Czajkowski, K., Foster, I.T., Kesselman, C., Sander, V., Tuecke, S.: Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In: Job Scheduling Strategies for Parallel Processing, 8th International Workshop, JSSPP 2002. (2002) 153–183