

**DIPARTIMENTO DI ELETTRONICA E INFORMAZIONE
POLITECNICO DI MILANO**



A user driven policy selection model

**Mariagrazia FUGINI, Pierluigi PLEBANI, Filippo RAMONI
Report n. 2006.65**

November 2006

A user driven policy selection model*

Mariagrazia Fugini, Pierluigi Plebani, and Filippo Ramoni

Politecnico di Milano, Dipartimento di Elettronica e Informazione
[fugini|plebani|ramoni]@elet.polimi.it

Abstract. During Web service discovery phase, the policy selection step must occur to retrieve not only the Web services able to perform the required functionalities, but also Web services able to ensure a given quality level.

Both service providers and service users define policies. The former list the supported service features and use a technical vocabulary (e.g., bandwidth, latency) whereas the latter identify the requirements using high level vocabulary closer to human users (e.g., speed).

The aim of this paper is twofold. On one hand, it introduces a model for expressing quality according to both applications and human users perspectives. Such a model, compliant with the WS-Policy framework, not only mediates between the application and human user perspectives, but is also capable of considering the different importance assigned by the user to quality dimensions. On the other hand, the paper introduces a policy selection model based on the adopted quality model. According to this approach, human users can express quality requirements according to a high level language; these requirements are then matched against low level specifications of the service quality.

1 Introduction

Web service selection plays a crucial role in Service Oriented Computing, since it is responsible for identifying which is the best Web service with respect to the user needs. Service selection can be performed by two kinds of users: applications and human beings. Even if both aim at selecting the best Web service with respect to their needs, the basis on which application and human users perform the selection are slightly different. Such a difference derives from the model that they adopt to describe the quality of a Web service. Technical parameters, such as throughput, bandwidth, latency, framerate could be comprehensible by applications and developers although less by final users: the latter have not skills to define the quality by technical parameters but they need a set of dimensions expressed through a high level specification, such as *video quality* or *audio quality*. Even the way such dimensions are measured may vary with respect to the kind of user. Applications might be interested on the exact number of frames per second or on the number of bits transmitted on a given time slot. On the contrary,

* Technical Report 2006.65 Dipartimento di Elettronica ed Informazione Politecnico di Milano Via Ponzio, 34/5 - 20133 Milano - Italy

human users are more familiar with discrete scales, such as *good*, *average*, and *worst*.

Nowadays, specifications available for expressing the quality of a Web service are mainly addressed to the applications and do not deal with the situation in which the service users are human beings. The aim of this paper is twofold. On one hand, it introduces a model for expressing quality of service according to both applications and human users perspectives. Such a model not only mediates between the application and human user perspectives, but is capable of considering the different importance assigned by the user to a given quality dimension. On the other hand, the paper introduces a policy selection model based on the adopted quality model. According to this selection model, a human user can express its requirements according to a high level language and its requirements are matched against the Web service quality specification expressed through a technical language. Both models are based on AHP (Analytical Hierarchical Process) developed by T.L. Saaty [?]. This is a decision-making technique that assigns to each alternative a score that represents the overall performance with respect to the different parameters.

The paper is structured as follows. Section 2 introduces the proposed model for describing and evaluating the quality of a Web service. Section 3 describes how the quality model fits in Ws-Policy for describing both the quality offered by a Web service and the quality desired by the related users. Finally, Section 4 introduces the selection policy model based on the elements previously described to figure out which is the best Web service among functionally equivalent ones, with respect to a quality request. A discussion on related work and some remarks conclude the paper.

2 Quality dimension model

In the literature ¹ several quality models have been proposed. In our opinion they are able to express the non functional properties of a service but they do not deal with the difference between user and service perspectives. In the same way, not all the quality dimensions have the same importance and such importance depends on both the application domain and the service users. The quality of Web service model we propose in this work aims at dealing with this aspect and it is based on three main elements: quality dimension model, quality of Web service definition model, and a quality evaluation model.

2.1 Quality dimension model

A *quality dimension*, also known as *quality parameter*, is a non-functional aspect related to an entity that we are describing. Thus, the quality dimension is close to the application domain we are taking into account and it can be directly measured or estimated starting from other dimensions. We identify two classes

¹ see <http://www.cs.uni-magdeburg.de/~rud/wsqs-links.html>

of quality dimensions, namely, *primitive* and *derived*. A *primitive quality dimensions* (*pqd* hereafter) is a directly measurable quality dimension and it is defined as follows:

$$pqd = \langle name, values \rangle \quad (1)$$

- *name*: uniquely identifies the quality dimension (e.g., framerate, bandwidth).
- *values*: defines the domain of values of the dimension. The domain can be either continue (e.g., 0..100) or discrete (e.g., high, medium, low).

On the other hand, a *derived quality dimension* (*dqd* hereafter) is not directly measurable but it depends on other quality dimensions:

$$dqd = \langle name, f(pqd_i, dqd_j) \rangle \quad i = 0..n, j = 0..m \quad (2)$$

- *name*: uniquely identifies the quality dimension.
- $f(pqd_i, dqd_j)$: the *dependency function* stating the influence of other quality sub-dimension (both *pqd* or *dqd*). The nature of the function may vary from a simple expression to a composite function.

2.2 Quality of Web Service definition model

The quality of Web service can be defined as the set of quality dimensions which express the non-functional aspects of a Web service. Due to the strong dependency of a quality dimension on the considered application domain, in our quality model we include an actor called *quality designer* who is in charge of collecting and organizing the relevant quality dimensions. Since the quality designer is a domain expert, he/she is also able to state if a quality dimension is primitive or derived. For example, in a trading on-line application domain, the bandwidth can be considered as a *pqd*. On the opposite, if we are considering a more technological domain such as inter-networking, *pqds* will be latency and jitter and they are involved in calculating the bandwidth which, in turn, will be a *dqd*. Given a *dqd*, the quality designer, also states if all the dependent dimensions defined in the related dependency function f must be considered. For example, the sound quality, a *dqd*, is usually described by two *pqds*: compression algorithms used and sampling frequency. Even if these two *pqds* composes the domain of f_{sound} , the quality designer can decide to take care only about the sampling since is not considered so peculiar the used algorithm.

According to (??), a *dqd* depends on both *pqds* and *dqds*, thus the work of the quality designer results in a tree named *quality tree* (QT) (see Figure ??):

$$\begin{aligned} QT &= \langle dqd_{QoS}, tree_node_k \rangle \quad k = 1..p \quad (3) \\ tree_node &= [\langle pqd \rangle \mid \langle dqd, v(pqd_i, dqd_j), w(pqd_i, dqd_j) \rangle] \\ &\quad i \leq n, j \leq m, domain(f_{dqd}) \supseteq domain(g) = domain(w) \end{aligned}$$

A QT refers to a given application domain and it includes and organizes all the relevant quality dimensions identified by the quality designer. Given a class

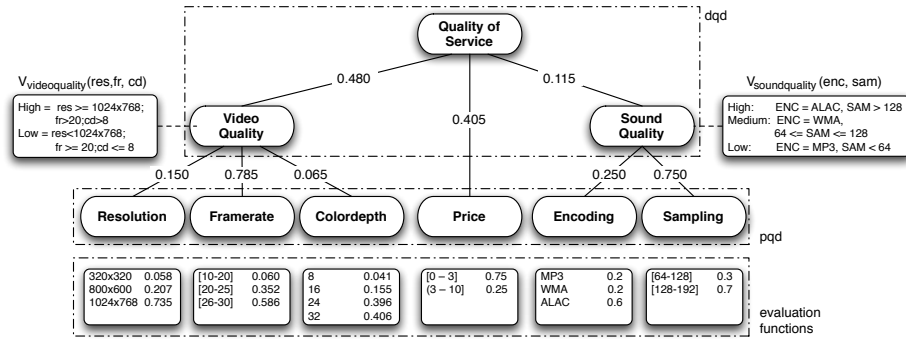


Fig. 1. Quality tree for video-on-demand Web services

of Web services (e.g.: video on demand, flight booking), both service providers and users will rely on the related QT tree to describe, namely, the offered and desired quality. So, the tree offers a common knowledge for reasoning about quality.

About the structure of a QT, the root is a *dqd* named *QoS*, leaves refers to *pqds*, and internal nodes are *dqds*. The function $v(pqd_i, dqd_j)$ is inspired by $f(pqd_i, dqd_j)$ and returns the value of a *dqd* with respect to the quality dimensions which the *dqd* depends on. The domain of f might contains the domain of v since the quality designer can decide to include in the quality tree only some of the dependent quality dimensions which usually define a given *dqd*.

It is worth noting that different position of a node inside the tree results in a different quality of service description. For example, in the tree shown in Figure ??, we suppose that the price is defined regardless of the other quality dimensions. Actually, it might happen that the price is directly influenced by both video and sound quality as well. In fact, better video and sound quality usually implies a higher price. In such a situation, the tree does not reflect this dependency as, on the contrary, the tree shown in Figure ??a does. Moreover, the price is the only quality dimension on which the overall quality of service directly depends on.

At the same time, it might happen that a given *dqd* is influenced by the same quality dimensions on which other *dqds* depend on. For example, as shown in Figure ??b, both video quality and price are influenced by the framerate. So, in this case, even the price is calculated starting from the framerate value and it implies that the framerate value variation reflects on both video quality and price evaluation.

In addition to the the function v , a tree node is also specified by a weight function w expressing the importance of the quality dimensions which the *dqd* depends on. For example, in Figure ?? we are stating that the video quality of video-on-demand Web service is defined by three main *dqds*: *video quality*, *sound quality*, and *price*. According to the weights, the video quality is the most im-

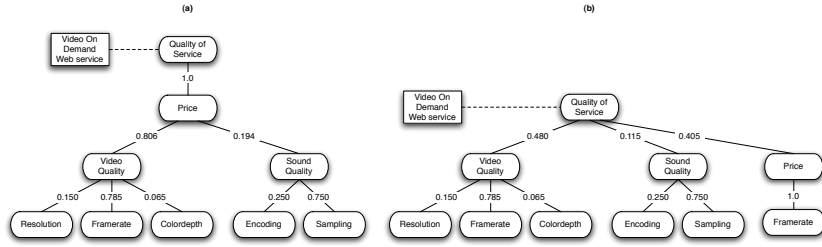


Fig. 2. Alternative QoS tree definition for a video on demand service

portant factor in evaluating the overall quality (0.480). Almost the same impact has the price (0.405), whereas the sound quality has the most limited impact (0.115). Recursively, the video quality is influenced by three main *pqds*: *resolution*, *framerate*, and *colordepth*. Here, the framerate has the most important influence on the video quality (0.785), and multiplying the weights assigned to the edge on the path from the root to the framerate node we obtain the influence of this *pqd* on the overall quality ($0.3768 = 0.480 * 0.785$).

The weight assignment is a quite critical activity. About this step, the quality designer has only one constraint: given a tree node, the sum of weights on the edges connecting to the direct children must be 1. Thus, the main problem for the quality designer is to realize how to distribute these weights. Considering the video quality example it is hard to quantify with a number in $[0..1]$ range, how much each sub-dimension influences the video quality.

Due to this difficulties, we adopt a qualitative approach where the quality designer only states if a sub-dimension is more influent than another one on the overall quality. For this reason, we adopt the *AHP* (*Analytic Hierarchy Process*) approach, developed by T.L. Saaty [?], to perform weight assignment. This 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 QT and proposes to user pairwise comparisons between sub-dimensions.

According to this approach, given a *dqd* the quality designer should fill tables similar to what shown in Table ???. The first column and the first row are populated with the name of the sub-dimensions influencing the given *dqd*. For each cell, the quality designer assigns a number in the $[\frac{1}{9}..9]$ range according to the meaning defined in Table ??? which is the usually adopted one in AHP.

For example, in Table ??? we state that *framerate* is extremely more important than then *colordepth* in defining the video quality (value 9). At the same time, the *resolution* has a moderated importance with respect to the *colordepth* (value 3). It is worth noting that the comparison matrix is always a reciprocal matrix, so its inferior part is reciprocal to the superior part and all the elements of the principal diagonal are 1. According to the AHP method, the principal eigenvector of the comparison matrix collects the weights we need. About our example, the

	Res	FR	CD
Resolution	1	$\frac{1}{7}$	3
Framerate	7	1	9
ColorDepth	$\frac{1}{3}$	$\frac{1}{9}$	1

Table 1. Comparison Matrix for VideoQuality dimension

a_{ij}	Definition
1	Equal importance
3	Moderate importance
5	Essential or strong importance
7	Demonstrated importance
9	Extreme importance
2, 4, 6, 8	Intermediate values (compromise)

Table 2. The Saaty Pairwise Combination Scale

eigenvector of the matrix in Table ?? is $\{0.150; 0.785; 0.065\}$. This motivates the values reported in QT of Figure ??.

Building the comparison matrix is a time consuming activity since it requires, for each dqd in QT, $\frac{n(n-1)}{2}$ comparisons where n is the number of sub-dimensions. There are some extensions, like the Harker variant [?], which increase usability, performance, and consistency by skipping some pairwise comparisons.

2.3 Quality evaluation model

The QT defined by the quality designer states the dependency among several quality dimensions and defines which are the possible values that such dimensions can assume. Relying only on the quality values is not enough to state which is the quality of a Web service. In some case, e.g., bandwidth, lower value means lower quality; in some other cases, e.g., price, higher value means lower quality. For this reason, nearby the QT, the quality designer also defines, for each quality dimensions in QT, an *evaluation function* which captures the quality trend with respect to the quality dimension value.

The evaluation function has different expressions with respect to the kind of quality dimension. For the pqd , the evaluation function ($e_{pqd}(values)$) is a punctual function required to state how a quality value is close or far to the best quality value. For instance, by definition the colordepth is a pqd described as follows:

$$\langle colordepth, \{8; 16; 24; 32\} \rangle \quad (4)$$

where 8 is the worst case, whereas 32 the best one. What is more interesting is to state which is the improvement in switching from 8 to 16 and from 16 to 32. In fact, given a quality dimension domain, the included values are simply state a total order relationship. So, the numerical gap between two values in a quality domain does not necessarily reflect a related quality gap. For instance, passing from a colordepth 8 to 16 has a bigger impact than passing from 16 to 24 even if the numerical gap is 8 in both cases. In turn, the quality improvement in switching from 24 to 32 will be less. Aim of the evaluation function is to explicitly declare the quality improvement switching from a value to another one.

Even in this case, AHP approach drives the definition of an evaluation function. Given a pqd , all its admissible values are compared to each other filling the

comparison matrix. As occurred in comparing several quality dimensions, even in this case the values in the comparison matrix must be in the $[\frac{1}{9}..9]$ range and such values can be obtained manually or semi-automatically. In the first case, all the comparison values are identified as done during the comparison among quality dimensions using a standard scale (see Table ??). In the semi-automatic approach, for each quality parameter, the quality designer associates a utility function (e.g., linear, logarithmic, exponential, sigmoid) reflecting the trend of quality increasing and suggesting reasonable values for the comparison matrix (see Table ??). In case of the colordepth, we assume that the utility function is logarithmic. The values in the comparison matrix are proportional to the gap between the values. If two values are the same then the value in the comparison matrix will be 1; on the opposite, the maximum gap will be associated to 9, whereas intermediate values are obtained proportionally (see Table ??). By calculating the eigenvector of the comparison matrix we obtain the elements of the evaluation function as reported in Figure ?? as well.

colordepth domain	log(values)
8	0.90
16	1.20
24	1.38
32	1.50

Table 3. Evaluation function for *resolution*

colordepth domain	8	16	24	32	eigenvector
8	1	5	7	9	0.041
16	$\frac{1}{5}$	1	3	5	0.155
24	$\frac{1}{7}$	$\frac{1}{3}$	1	3	0.396
32	$\frac{1}{9}$	$\frac{1}{5}$	$\frac{1}{3}$	1	0.406

Table 4. Pairwise comparisons for *resolution*

Considering the *dqd*, the evaluation function ($e_{pqd}(QT, pqd_i, dqd_j)$) is a linear combination of the quality dimensions which influence such a *dqd* according to the considered QT (i.e., $domain(e) = domain(v_{pqd})$). In particular, since the influencing quality dimensions can be both primitive or derived, the evaluation function of a *dqd* will be:

$$\begin{aligned}
 e_{dqd}(QT, pqd_i, dqd_j) = & \sum_{i=0..n} e_{pqd}(pqd_i.values) * w(pqd_i) + \\
 & + \sum_{j=0..m} e_{dqd}(QT, domain(g_{dqd_i})) * w(dqd_i)
 \end{aligned} \tag{5}$$

3 Policy model

A policy is a document stating the requirements or the offering of a Web service. Following the quality dimension model, a policy document collects a set of quality dimensions and defines the admissible values for each of them. The service

provider attaches a policy document, named *Service policy*, to a Web service to define which is the offered quality. On the opposite, the service user relies on a policy called *User policy* to list the requirements for a Web service.

3.1 Service Policy

Service policy includes only *pqds* and it is defined when the service provider advertises the related Web service (for instance, when publishing its specification on a UDDI Registry). Operatively, the service policy definition process is composed by three main steps:

1. First, we assume that a QT associated to the kind of advertising Web service exists. Considering the video on demand scenario, we assume to rely on the QT of Figure ??.
2. Secondly, given a quality tree, the service provider will extract the list of included *pqds*, hence the tree leaves are selected. Such a list represents the set of quality dimensions that the provider can include in a policy (e.g., resolution, framerate, colordepth, encoding, sampling, price).
3. Finally, for each selected *pqd*, the service provider defines an admissible value range with respect to what the Web service can ensure. In fact, a *pqd* is defined regardless of the policy which it might be included in. When a service provider decides to include a *pqd* in a policy describing one of its service, then he must define which is values subset characterizing in general the *pqd* which still valid for the Web service. For example, nowadays it is reasonable to consider that the framerate can assume values included in [10..30fps]. The service provider, according to the implementation of its service, can include in a policy the *pqd* framerate with values between [20..25fps].

Ws-Policy framework [?] inspires the organization of the policy document. Ws-Policy, the main specification of this framework, is responsible to organize assertions according to two main operators: **ExactlyOne** and **All**. An assertion can be defined as a specialization of a quality dimension with a restricted admissible value set as done during the step 3.

$$a(pqd) = \langle pqd.name, values \subseteq pqd.values \rangle \quad (6)$$

Given a set of assertions, applying the **ExactlyOne** operator, only one assertion must hold at the same time, whereas, applying the **All** operator, each assertion must hold. As described in the Ws-Policy framework specification **All** and **ExactlyOne** operators can be associated to, namely, AND and XOR boolean operator, so that a Ws-Policy document be considered as a boolean expression where the terms are the policy assertions. By exploiting such an interpretation, as suggested in [?], we can always obtain the Disjunctive Normal Form (DNF) from the boolean expression underlying a Ws-Policy document where an **ExactlyOne** operator includes a set of **All** operators. By means of this DNF expression, we can explicitly identify the set of policy alternatives that can indicate a set of

non-functional requirements. An alternative $A_k(QT)$ corresponds to one of the `ExactlyOne` blocks so:

$$A_k(QT) = \bigwedge_{pqd_i \in QT} a(pqd_i) \quad (7)$$

It is worth noting that in a DNF form, the outer operator is `All`, so, only one of the included assertions can hold. According to this approach a policy P can be defined as follows:

$$P(QT) = \bigoplus_{k=1..l} A_k(QT) \quad (8)$$

where the following conditions holds:

$$\exists i, j | A_i(QT), A_j(QT) \in P(QT) \wedge A_i(QT) = A_j(QT) \quad (9)$$

$$A_i(QT) = A_j(QT) \Leftrightarrow \forall a_r \in A_i(QT) \exists a_s \in A_j(QT) | \quad (10)$$

$$a_r.name = a_s.name \wedge a_r.values = a_s.values$$

In this way, a policy includes a set of distinct alternatives which, in turn, include the same assertion set. So, two alternatives in the same policy differ if exists at least one assertion with the same name but a different value range.

3.2 User Policy

Users follow almost the same process introduced for the service policy to build the user policy. Once a user has identified the QT collecting all the relevant quality dimensions (as the service provider also does), he can express his requirements defining the admissible values for both pqd and dqd . In some cases, user expresses numeric ranges (`framerate=[22..25fps]`), in other cases he selects a set of higher level values (`video quality = high`). In this way, the user can both express his requirements at different levels of details.

The requirements expressed by the user are processed to be transformed into a user policy compliant with (??) which includes only pqd . This transformation process is quite simple since it exploits the relationships expressed by the QT. For example, the requirements `framerate = [22..25fps]` and `video quality = high` will be converted in: `res ≥ 1024x768; 22fps ≤ fr ≤ 25fps; cd > 8`. For each pqd , the user also states if the value range must be totally covered by the service offerings, or it is considered fine that the service provider is able to offer even only a subset of it. For this reason we distinguish between *mandatory* and *non-mandatory* quality dimension requests expressed in a user policy by a flag attached to each pqd .

In addition, a user can also customize the weights defined in the QT. For this purpose, the user imports a copy of the QT and modifies the weights according to its own preferences. For example, the price can be differently perceived by different users. Somebody, in fact, might desire a good quality regardless of the

price. At the same time, somebody else might prefer to save money accepting a lower video and sound quality. The modified version of the quality tree according to the user preferences is called User Quality Tree (*UQT*).

4 Policy selection model

Given a user request and a set of Web service policies, the policy selection is in charge of figuring out the best Web service policy with respect to the user preferences. The policy selection considers Web services of the same type so, the related quality tree (QT) can be obtained by the quality designer for the given application domain.

According to our quality model, we assume that service providers and users have documents specifying quality constraints and preferences with respect to the same quality tree. In detail, at the provider side, we have a service policy document $SP(QT)$ (SP hereafter) which specifies the quality of service with respect to several configurations, i.e. alternatives. At user side we have both a user policy document $UP(QT)$ and a customized quality tree (UQT). It is worth reminding that a policy (both user and service) are organized according to (??), and all the assertions included in an alternative will refer to *pqd*.

The selection process starts when an input of type SP , UP and UQT ² is received. The process output is a revised policy (RP) where the alternatives included in SP are sorted from the alternative which best matches the user requirements to the worst one. Actually, user also submits his functional requirements but the issues related to the retrieval mechanisms based on the functional perspective are out of the scope of the present paper [?]. So, we suppose after the user expressed its requirements, the system has already collected the set of service policies associated with the set of Web services providing the required functionalities.

The selection process is composed by two main steps: *Satisfiability evaluation* and *Alternative ranking*. Figure ?? exemplifies the process considering the video-on-demand scenario.

4.1 Satisfiability evaluation

Given a $A_i \in SP$, the satisfiability evaluation aims at stating if

$$\forall u_j \in UP, \exists s_i \in A_i \mid s_i \text{ satisfies } u_j \quad (11)$$

The operator *satisfies* considers both the name and the values of a quality dimensions. About the former:

² For the sake of simplicity, we only describe the matching between a UP and a single SP where the latter expresses several configurations. In addition, we assume that the user does not modify the QT so $UQT = QT$. The general scenario where a set of SP s are considered and $UQT \neq QT$ can be simply derived.

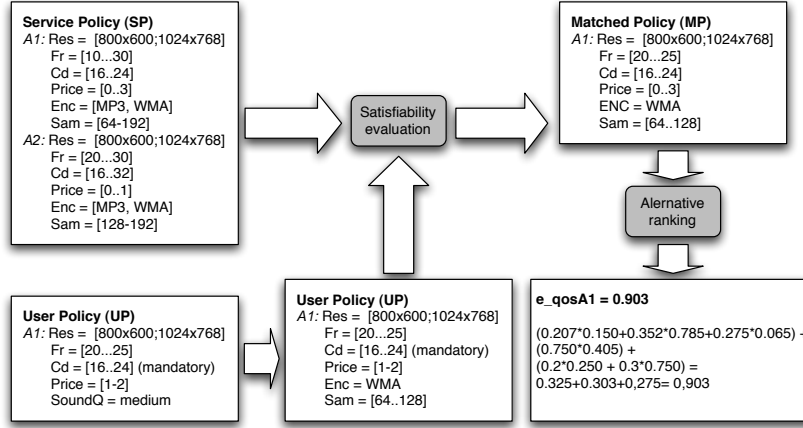


Fig. 3. Example of SP, UP and quality evaluation

$$s_i \text{ satisfies } u_j \Rightarrow s_i.name = u_j.name \quad (12)$$

Roughly speaking, during this activity, the selection process verifies that, for all the service requests expressed in UP , there exists at least one of the service offering assertions which satisfies such a request. This means that all the quality dimensions included in UP must be included in SP as well. If at least one of the quality dimensions in UP is not satisfied, then the process considers the alternative A_i as not compliant.

At the opposite, it might happen that a quality dimension included in the SP could not be considered in the UP . In this case, the process continues since the user might be unaware about a quality dimension that the Web service offers.

The operator *satisfies* also considers the values describing an assertion. About this analysis, we first need to distinguish between mandatory and non-mandatory value ranges. If a value range is mandatory then following definition holds:

$$s_i \text{ satisfies } u_j \Rightarrow s_i.value \supseteq u_j.value \quad (13)$$

Instead, if a user defines a non-mandatory value range then:

$$s_i \text{ satisfies } u_j \Rightarrow s_i.value \cap u_j.value \neq \emptyset \quad (14)$$

The satisfiability evaluation results in a MP (matched policy), a revised version of SP where the structure remains the same of SP and the value ranges are redefined according to the user request:

$$\begin{aligned}
MP = \bigoplus A_m \mid & (\forall A_m, \exists A_k \in SP \mid \\
& (\forall s_i \in A_k, \exists m_i \in A_m \mid \\
& (m_i.name = s_i.name = u_j.name \wedge \\
& m_i.values = s_i.values \cap u_j.values)))
\end{aligned} \tag{15}$$

4.2 Alternative ranking

The second step of the selection process has to sort the alternatives included in MP taking into account the importance assigned by the user to the quality dimensions. So, the inputs of the alternative ranking phase are both MP and UQT whereas the output is the final policy RP (Ranked Policy).

Similarly to what done during the satisfiability evaluation, RP has the same structure of MP and is defined as follows:

$$\begin{aligned}
RP = \bigoplus A_r \mid & (\forall A_r, \exists A_m \in MP \mid A_r = A_m) \wedge \\
& (\forall A_i, A_j \in RP, i < j \Rightarrow \\
& e_{qosA_i}(UQT, domain(g_{qosA_i})) \geq e_{qosA_j}(UQT, domain(g_{qosA_j})))
\end{aligned} \tag{16}$$

The quality of an alternative that we need to rank the alternatives in a policy is obtained using the evaluation functions of the assertions composing the alternatives. An alternative in MP , in fact, is expressed in terms of assertion related to pqd which, in turns, are the leaves of the quality tree associated to the alternative as well. Actually, during the quality calculation we do not consider the original quality tree but UQT , i.e., the version the user customized.

$$quality(A, UQT) = e_{qos}(pqd_i) \quad pqd_i.name \in UQT = a_k.name \in A \tag{17}$$

5 Related work

A complete service description is an important requirement for users who aim at searching the most suitable Web services. Besides the functional description, where WSDL represents the most adopted specification, even non-functional specification are considered. WSOL [?], WSLA [?], for instance provide some description model which our work can use to express pqd . Focusing on selection process, in [?] the dynamics selection of the services is discussed proposing a solution based on agents, using the Web Services Agent Framework (WSAF), that includes an ontology for the QoS and a *ad-hoc* language to specify quality. The proposed approach only evaluate services with feedback assigned from the user that have already used the service, and does not consider the actual users' needs. In [?], considering Web Service as highly configurable objects and that the simple attribute-value pairs are insufficient to describe offers and requests, two requirements are identified for the automatic selection and the negotiation of services: the preferences information and the cardinal preferences. The proposed

utility theory uses utility functions to estimate every parameter. In this work the authors do not focus on the dynamic creation of the dimensions and don't suggest how users could personalize quality dimensions.

In this paper the quality designer is inspired by the community's role in the QoS defining process as it is developed in [?]. Here, different communities collaborate to define QoS dimensions; our model could be applied in each community of the model (device, network and service community).

In our work we have adopted *WS-Policy* as policy language; other languages, such as *Features and Properties (F&P)*, are also available. Different work, like [?], show the substantially equivalence between the two languages, finding the differences at the syntax level. Both languages use tags to represent metadata, but F&P is based on URL to express components and property, while WS-Policy use XML vocabulary and Qnames.

6 Concluding remarks

In this work we have proposed an approach for selecting Web services by analyzing the offered quality. A quality definition and evaluation model are introduced to allow both Web service providers and users to specify, namely, the offered and desired quality. Such models also deal with the different levels of details in expressing quality by these two actors. Based on this model, the selection process we propose is capable of automatically matching user and provider policies and ranking several quality alternatives to identify the best Web service. A prototype implementing our approach is under development.

References

1. G. Daniels. Comparing Features & Properties and WS-Policy. *W3C Workshop on Constraints and Capabilities for Web Services*, 2004.
2. S. Baja et Al. Web Services Policy Framework (WS-Policy). <ftp://www6.software.ibm.com/developer/library/ws-policy.pdf>, 2004.
3. P.T. Harker. Incomplete Pairwise Comparisons in the Analytic Hierarchy Process. *Mathematical Modelling 9/11*, 1987.
4. A. Keller and H. Ludwig. 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, May 2002.
5. S. Lamparter and S. Agarwal. Specification of Policies for Web Service Negotiations. *Semantic Web and Policy Workshop, Galway*, November 2005.
6. C. Marchetti, B. Pernici, and P. Plebani. A Quality Model for Multichannel Adaptive Information System. accepted for 13th International World Wide Web Conference, WWW04, New York, Usa, May 2004.
7. E. M. Maximilien and M. P. Singh. A Framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Computing*, September-October 2004.
8. T. Mikalsen, N. K. Mukhi, P. Plebani, and I. Silva-Lepe. Supporting Policy-driven behaviors in Web services: Experiences and Issues. *ICSOC-04*, 2004.
9. P. Plebani. *Searching and Invoking e-Services in Multichannel Information Systems*. PhD thesis, Politecnico di Milano, 2005.

10. T. L. Saaty. *The Analytic Hierarchy Process*. Mc Graw Hill, New York, 1980.
11. V. Tasic, K. Patel, and B. Pagurek. WSOL - Web Service Offerings Language. In *Web Services, E-Business and the Semantic Web, CAiSE 2002 International Workshop (WES 2002)*, Toronto, Canada, May 2002.