# A Conditional Lexicographic Approach for the Elicitation of QoS Preferences

Raluca Iordache and Florica Moldoveanu

University "POLITEHNICA" of Bucharest, Romania
riordache@hotmail.com, fm@cs.pub.ro

**Abstract.** In a service-oriented environment, clients can usually choose between several web services offering the same functionality. The web service selection can be automated by allowing clients to specify non-functional requirements such as quality of service. Clients should also be able to indicate how to make tradeoffs when some of these requirements cannot be met. The ability to capture tradeoff preferences is critical for selecting the best fitting web service. In this paper, we propose a method of expressing non-functional preferences, which requires minimal effort on the part of the clients, but offers great flexibility in managing tradeoffs. This method leads to a simple algorithm for selecting web services, which does not require sophisticated multicriteria decision techniques.

**Keywords:** QoS preferences, multicriteria decision making, service selection.

## 1 Introduction

The emergence of services brought the world of computing in front of a new level of abstraction that is closer to the way humans naturally think and interact with their surroundings [1]. In real life, people make use of particular services, after selecting from the available alternatives the ones suiting best their requirements. In service-oriented environments, the existence of numerous web services offering the same functionality needed for a given task leaves the application designer with several candidates to choose from. At this point, analyzing the quality of the alternatives starts playing a fundamental role in the service selection. The non-functional characteristics of a web service, such as availability, cost, response time, or the supported security protocols define the Quality of Service (QoS) concept.

Although considerable research has been done in the recent years, there is currently no widely accepted approach for the QoS-aware selection of web services. This is mainly due to the various issues that have to be addressed in order to provide a complete solution. These issues include the design of suitable frameworks and architectures [2][3], which should provide ontologies for the formal specification of QoS metrics [4][5], methods of obtaining current metric values [6] and algorithms that select the best web service based on user-specified QoS criteria [7].

The ability of clients to express their QoS expectations plays a crucial role in the selection of the most suitable web service. While hard constraints are relatively easy to formulate, there is no standard way to deal with soft constraints that should reflect client's preferences in situations where no web service is capable of satisfying all QoS requirements. Most approaches are based on specifying priorities or associating weights to the different QoS dimensions. The drawback of these methods is that they cannot accurately capture user's preferences. On the other hand, more elaborate ways of specifying QoS preferences usually require considerable effort on the part of the clients. This complexity brings the risk that users don't understand the method or they are not willing to spend so much time expressing their preferences.

In this paper, we propose a conditional lexicographic method of articulating non-functional preferences, which offers great flexibility, while being easy to use and understand. It is based on the way people reason about their preferences, thus fostering its acceptance by users. This method leads to a simple algorithm for selecting web services, which does not require sophisticated multi-criteria decision techniques.

The rest of this paper is organized as follows: Section 2 presents the addressed problem of expressing the user's preferences. Section 3 outlines the related work of service selection based on multicriteria techniques. Section 4 introduces our conditional lexicographic approach illustrated by a case study of a data visualisation service. The last section concludes the paper and outlines future work directions.

## 2 The Problem of Expressing Preferences

Preference models can be found in various areas like psychology, mathematics, philosophical literature, in economics and game theory, in operations research and decision analysis and in various disciplines of computer science. The choices that we make are guided by our preferences. Understanding preference handling is relevant when attempting to build systems that make choices on behalf of users [8].

Of particular interest for the domain of QoS-aware service selection are the fields of multicriteria decision analysis and in particular of multiobjective optimization or Pareto optimization. Multiobjective optimization problems can also be found in various areas where optimal decisions involve tradeoffs between multiple (possibly conflicting) objectives. A Pareto optimal solution is a solution for which it is impossible to improve one objective without worsening another one. Multiobjective optimization uses a priori or a posteriori approaches, depending on the moment when the decision maker's preferences are articulated.

The best known and simplest method for preference articulation is the weighted sum method. The method uses weight values supplied by the user to describe the importance of the objectives. One drawback of this method is that the weights must both compensate for differences in objective function magnitudes and provide a value corresponding to the relative importance of an objective. Another drawback is that it is not able to find certain solutions in the case of a non-convex Pareto curve. The authors of [9] conclude that the weighted sum method "is fundamentally incapable of incorporating complex preference information".

Lexicographic preferences is another simple method used for modeling rational decision behavior. Preferences are defined by a lexical ordering, which leads to a strict ranking. While being very easy to use, lexicographic preferences have the major drawback of being non-compensatory. An extension of this method is lexicographic semiorder, where a tradeoff is addressed in situations where there is a significant improvement in one objective that can compensate an arbitrarily small loss in the most important objective. An alternative x is considered better than an alternative y if the first criterion that distinguishes between x and y ranks x higher than y by an amount exceeding a fixed threshold [11]. The advantage of this method is that it ensures that a solution that is slightly better on the most important objective but a lot worse on the other objectives will not be selected.

## 3 Existing Methods and Approaches

Most work on QoS-aware service selection uses successive evaluation of different, non functional aspects in order to attribute a general "level of quality" to a service [3]. Usually the selection of the service offering the best functionality is based on either a single evaluation criterion or on a weighted sum of several quantitative evaluation criteria. These approaches have in practice major disadvantages because in most of the cases a single criterion is not enough for defining the user's QoS requirements and the weighted sum of the criteria lead to compensation problems and so to inadequate results. Many QoS attributes are qualitative parameters and cannot be used on a weighted sum evaluation. By using quantification and assigning values to the qualitative data, these parameters can be transformed in quantitative parameters. This is done usually by defining a measurement scale and then associating to each level of the scale a numerical value. For example the scale can be defined as 1-5, with 1 being very high and 5 being very low and these numbers are associated to the different qualitative attributes, by their relevance.

In practice such a quantification approach doesn't show good results in the QoS-aware service selection, and the challenge of obtaining good results lies in managing tradeoffs among QoS expectations in situations in which service requesters specify quality levels that cannot be simultaneously met.

This key problem of managing tradeoffs of the QoS preferences is addressed by [12]. The authors propose a QoS model for describing the QoS dimensions (requirements and preferences) of the service requester and the service provider. They are using fuzzy multicriteria decision analysis (MCDA) for comparing the models and ranking the competing services according to the values of their characteristics. The model describing the QoS properties is based on OMG's UML QoS Framework metamodel. MCDA methods allow the defining of weights related to criteria and also weights to interactions between criteria.

The authors of [13] propose an extension of the Web Service architecture by adding to the UDDI registry a new component called Multicriteria Evaluation Component (MEC) used for the multicriteria evaluation. This evaluation is based on a Web version of IRIS (Interactive Robustness Analysis and Parameters Inference for multicriteria Sorting Problems), which uses the ELECTRE TRI method.

Dealing with preferences, their priorities and possible tradeoffs between them has been addressed by [11] using a model of lexicographic semiorder. The work is addressing decision making based on lexicographic heuristics and ranking in order to compare a pair of alternatives.

## 4 The Conditional Lexicographic Approach

As mentioned before, QoS expectations can take the form of hard and soft constraints. While all hard constraints must be satisfied in order for a web service to be selected, soft constraints represent rather desirable characteristics of the chosen service. If no web service meets all soft constraints, users should have the possibility to express their tradeoff preferences, in order to allow the dynamic selection of services.

Our approach to articulate the QoS preferences is based on the observation that, when trying to find a set of rules allowing them to choose between several alternatives, people start by ranking their preferences, in accordance with their perceived importance. This action is equivalent to imposing a lexicographic order on the different criteria that have to be considered. In most situations, using such a strict hierarchy is not sufficient to capture people's real preferences. In this case, people usually introduce additional rules that change the criteria priorities when some specific condition is met.

We propose a method to establish a total order on the set of existing web service alternatives, by attaching conditions to lexicographic preferences and we introduce a preference specification language that can be used for authoring QoS preferences.

We illustrate our approach and the use of its associated specification language by considering a hypothetical company that offers data visualization services. One of these services is the generation of charts based on data sets. Instead of performing itself such tasks, the company delegates them to other business partners, which offer web services for chart generation. A service selection broker chooses the most suitable web service, based on QoS requirements formulated by clients. The chart generation web services are characterized by domain-independent QoS attributes (e.g., availability, response time) and domain-specific ones (e.g., chart type, cost per chart, number of colors, image resolution).

As client of the hypothetical company we consider a data acquisition system, which regularly sends charts depicting the state of an industrial process to a 1280 x 720 monitor capable of displaying 65536 colors. The image displayed on the monitor can be updated only at fixed intervals of 5 seconds. If a new chart is not available at the end of a 5 seconds interval, the monitor update is postponed until the next end of a 5 seconds interval.

Our preference specification language allows specifying both constraints and preferences. Constraints are declared as a list of comma separated boolean conditions that must be satisfied by the service. They are enclosed in a *constraints* block, as shown in Fig. 1.

```
constraints {
   chartType = "time series",
   cost < 10,
   availability > 0.95,
   imageResolution = "1280x720",
   responseTime < 10
}
```

**Fig. 1.** Hard constraints specification

The order of constraint conditions is irrelevant, but order plays a key role in the articulation of QoS preferences. For the beginning, we consider that the client provides a strict ranking of preferences. This is expressed in our specification language by using a *preferences* block that includes the comma separated list of relevant QoS attributes in the order of their importance, as shown in Fig. 2.

```
preferences {
   cost,
   availability : high,
   responseTime,
   colors : high
}
```

**Fig. 2.** A simplistic specification of preferences

For each QoS attribute the client should indicate the direction associated with better values. This piece of information appears after the attribute name, separated by a colon. Possible values for direction are *low* and *high*, where *low* is the default direction and can be omitted.

In the example above, *cost* is the most important QoS attribute, and services with a lower cost are considered better. However, this specification of preferences does not accurately capture client's preferences. A first problem is that selecting a web service with a response time greater than 5 seconds would result in skipping an update of the monitor. This is a serious issue, and such a scenario should be prevented even if this leads to a higher cost. The *responseTime* attribute should be ranked higher only when exactly one of the two web services compared has a value higher than 5 seconds for this attribute. If, for example, both web services considered are able to provide the chart in less than 5 seconds, the problem of missing an update does no longer exist and *responseTime* does not need a higher ranking. Conversely, if both web services compared have a *responseTime* higher than 5 seconds, an update of the monitor will be inevitably skipped, and the actual value of this attribute is no longer of critical importance.

Another problem arises when at least one of the web services compared provides a number of colors less than 65536. Since this will lead to a loss of quality, the client may want to increase in such situations the importance of the *colors* attribute. If both web services provide a number of colors higher than 65536, the *colors* attribute becomes irrelevant, because the difference in quality cannot be detected on the available monitor.

Finally, a small difference in the values of the *cost* attribute should be ignored if the selection of the slightly more expensive web service leads to a better color quality.

In order to be able to articulate preferences for scenarios like the one above, our specification language provides four unary preference operators, which are shown in Table 1.

**Table 1.** Preference operators

| **Preference operator** | **Meaning** |
|---|---|
| **AT_LEAST_ONE***(condition) | condition($service_1$) OR condition($service_2$) |
| **EXACTLY_ONE**(condition) | condition($service_1$) XOR condition($service_2$) |
| **ALL**(condition) | condition($service_1$) AND condition($service_2$) |
| **DIFF**(attribute) | $\lvert service_1.attribute - service_2.attribute \rvert$ |

*default operator (can be omitted)

The first three operators take as argument a boolean formula, which usually involves one or more QoS attributes. The formula is evaluated twice, once for each of the web services to be compared. The two resulting boolean values are passed as arguments to the boolean operator (OR, XOR, or AND) associated with the given preference operator, in order to obtain the return value.

The preference operator DIFF takes as argument a QoS attribute and returns the modulus of the difference of its corresponding values from the two web services compared. Our specification language also allows the definition of virtual QoS attributes, which will be treated as genuine QoS attributes by the preference operators. This can be done by means of the *def* directive, as seen in the example below:

```
def colorDepth = log2(colors)
```

In the remainder of this paper, we use the term *preference rule* to denote an entry in the *preferences* block. As already seen, a preference rule has three components: an optional *condition*, an *attribute* indicating the QoS dimension used in comparisons and a *direction* flag stating which values should be considered better. In our specification language, the preferences corresponding to the above described scenario can be articulated as shown in Fig. 3:

```
preferences {
   [EXACTLY_ONE(responseTime > 5)] responseTime,
   [DIFF(cost) > 2] cost,
   [colors < 65536] colors : high,
   availability : high,
   responseTime,
   colors : high
}
```

**Fig. 3.** A more elaborate specification of preferences

The condition part of the third preference rule in the above *preferences* block (i.e., [colors < 65536]) does not explicitly specify a preference operator, which means it uses the default operator AT_LEAST_ONE.

The specification language can deal with situations where people are not fully aware of their preferences. When users notice that the current rules do not accurately capture their preferences, they can simply add a new conditional rule, thus incrementally improving the preference specification.

In what follows, we use the notation $s_1 \succ s_2$ to indicate that the web service $s_1$ is preferred to the web service $s_2$, and the notation $s_1 \sim s_2$ to indicate that the service $s_1$ is indifferent to the web service $s_2$ (i.e., $s_1$ and $s_2$ are equally preferred). Additionally, we introduce the notation $s_1 \underset{k}{\succ} s_2$ to indicate that the web service $s_1$ is preferred to the web service $s_2$ and that the preference rule $k$ has been decisive in establishing this relationship. We also introduce the complementary operators $\prec$ and $\underset{k}{\prec}$, defined by the following relations:

$s_1 \prec s_2$, iff $s_2 \succ s_1$

$s_1 \underset{k}{\prec} s_2$, iff $s_2 \underset{k}{\succ} s_1$

An algorithm for comparing two web services based on the preferences expressed using our conditional lexicographic approach is shown in Fig. 4.

The algorithm examines all entries in the *preferences* block in the order in which they appear (line 2). If the current preference rule has no attached condition or the attached condition evaluates to true (line 6), the values corresponding to the attribute specified by this entry are compared (line 7). The *compare* function returns a numerical value that is positive if the first argument is better, negative if the second argument is better and 0 if the arguments are equal (see pseudocode in Fig. 5). If the attribute values are not equal (line 8), the algorithm returns a tuple containing the result of the current comparison and the index of the preference rule that has been decisive in establishing the preference relationship (line 9). Otherwise, the algorithm continues its execution with the next preference rule. A null return value (line 13) indicates an indifference relation between the two web services, while a not-null tuple identifies a relation of type $\underset{k}{\prec}$ or $\underset{k}{\succ}$ between them.

```
function compareServices(service1, service2, preferences)
   for i $\leftarrow$ 1 .. length(preferences) do
      cond $\leftarrow$ preferences[i].condition
      attr $\leftarrow$ preferences[i].attribute
      dir $\leftarrow$ preferences[i].direction
      if cond = null OR cond(service1, service2) = true then
         result $\leftarrow$ compare(service1.attr, service2.attr, dir)
         if result $\neq$ 0 then
            return {result, i}
         end if
      end if
   end for
   return null
end function
```

**Fig. 4.** Pairwise comparison of two web services

```
function compare(attr1, attr2, dir)
   if attr1 = attr2 then
      result $\leftarrow$ 0
   else if attr1 <attr2 then
      result $\leftarrow$ 1
   else
      result $\leftarrow$ -1
   end if
   if dir = high then
      result $\leftarrow$ -result
   end if
   return result
end function
```

**Fig. 5.** Pairwise comparison of QoS attribute values

In a series of experiments, Tversky [10] has shown that people have sometimes intransitive preferences. Therefore, being able to capture such preferences is an important feature of our specification language. However, a consequence of allowing intransitive preferences is that the pairwise comparison of all web service alternatives is in general not sufficient to impose a total order on these services. In order to illustrate this, we use a simplified version of the preferences specified in Fig. 3. As shown in Fig. 6, the simplified version does no longer involve the QoS attribute *availability*. Therefore, this attribute is no longer relevant for the web service ranking. Although the simplified specification used for exemplification is unrealistic, it is easier to analyze and it helps us highlight the intransitivity issues. (The preference rule indexes appearing at the left side of the figure are only informative and are not part of the preference specification.)

We consider a set of 5 web service alternatives ($WS_1$ through $WS_5$) with the relevant QoS attribute values specified in Table 2.

```
     preferences {
1.       [EXACTLY_ONE(responseTime > 5)] responseTime,
2.       [DIFF(cost) > 2] cost,
3.       [colors < 65536] colors : high,
4.       responseTime
     }
```

**Fig. 6.** A simplified specification of preferences used for exemplification

**Table 2.** Relevant QoS attribute values

| | $WS_1$ | $WS_2$ | $WS_3$ | $WS_4$ | $WS_5$ |
|---|---|---|---|---|---|
| responseTime | 7.0 | 7.0 | 5.5 | 4.5 | 7.5 |
| cost | 4.0 | 5.0 | 6.5 | 8.0 | 7.5 |
| colors | 256 | 256 | 256 | 65536 | 65536 |

The relations identified by the pairwise comparison of the 5 web services considered in our example are depicted in Table 3, where header notations use the format `i / j` to indicate that the corresponding symbol in the line below represents the preference relation between the web services $WS_i$ and $WS_j$.

**Table 3.** Pairwise comparison of the 5 web services

| 1/2 | 1/3 | 1/4 | 1/5 | 2/3 | 2/4 | 2/5 | 3/4 | 3/5 | 4/5 |
|---|---|---|---|---|---|---|---|---|---|
| $\sim$ | $\succ_2$ | $\prec_1$ | $\succ_2$ | $\prec_4$ | $\prec_1$ | $\succ_2$ | $\prec_1$ | $\prec_3$ | $\succ_1$ |

Several cases of intransitivity of preferences can be observed in the above table. A first example is given by the following relations:

$$WS_1 \succ WS_3 \succ WS_2$$
$$WS_1 \sim WS_2$$

Although $WS_1$ is indifferent to $WS_2$, $WS_1$ is preferred to $WS_3$, while $WS_2$ is not preferred to $WS_3$.

Another example is the rock-paper-scissors relationship induced by:

$$WS_2 \prec WS_3 \prec WS_5$$
$$WS_5 \prec WS_2$$

In order to obtain a total order on the set of web service alternatives, we attach to each web service $i$ a score vector of integer values: $V_i \in \mathbb{N}^{r+1}$, where $r$ is the number of preference rules. The algorithm used to compute the score vectors is presented in Fig. 7, where $n$ denotes the number of web service alternatives.

```
procedure createScoreVectors()
  for i ← 1 .. n do
    for k ← 1 .. r do
      $V_i^k$ ← number of times service $WS_i$ is preferred to another
            web service due to decisive rule k (i.e., due to a $\underset{k}{\succ}$ relation).
    end for
    $V_i^{r+1}$ ← number of times service $WS_i$ is indifferent to another web service.
  end for
end procedure
```

**Fig. 7.** Procedure to create the score vectors

For the 5 web service alternatives considered in our example, the corresponding score vectors computed with the above algorithm are presented in Fig. 8.

| | $\underset{1}{\succ}$ | $\underset{2}{\succ}$ | $\underset{3}{\succ}$ | $\underset{4}{\succ}$ | $\sim$ |
|---|---|---|---|---|---|
| $WS_1$ | 0 | 2 | 0 | 0 | 1 |
| $WS_2$ | 0 | 1 | 0 | 0 | 1 |
| $WS_3$ | 0 | 0 | 0 | 1 | 0 |
| $WS_4$ | 3 | 0 | 0 | 0 | 0 |
| $WS_5$ | 0 | 0 | 1 | 0 | 0 |

**Fig. 8.** Score vectors of the 5 web service alternatives

Using the score vectors, we are able to provide an algorithm for the ranking of web service alternatives. This algorithm is based on the function *compareScores*, described in pseudocode in Fig. 9. Again, $r$ is used to denote the number of preference rules. The function takes as arguments two score vectors and returns a numerical value that is positive if the web service corresponding to the first score vector is preferred, negative if the web service corresponding to the second score vector is preferred and 0 if the corresponding web services are indifferent to each other.

For each of the two corresponding web services, the function computes the number of times it has been preferred to other web services (lines 2, 3). This computation does not take into account the number of times a web service has been found to be indifferent to another one (hence the sum is taken up to the value $r$, not $r + 1$).

If the previously computed values $count_1$ and $count_2$ are not equal (line 4), the web service with the higher value is chosen as the better one (line 5).

Otherwise, the algorithm scans each position in the score vectors (line 7) and if it finds different values, the web service corresponding to the higher value is chosen as the better one (lines 8-10). The scanning of the values in the vector scores starts with the position corresponding to the first preference rule, because this is considered the most important one, and it ends with the position

```
 1. function compareScores($V_1$, $V_2$)
 2.     $count_1 \leftarrow \sum_{i=1}^{r} V_1^i$
 3.     $count_2 \leftarrow \sum_{i=1}^{r} V_2^i$
 4.     if $count_1 \neq count_2$ then
 5.         return $count_1 - count_2$
 6.     end if
 7.     for i ← 1 .. r + 1 do
 8.         if $V_1^i \neq V_2^i$ then
 9.             return $V_1^i - V_2^i$
10.         end if
11.     end for
12.     return 0
13. end function
```

**Fig. 9.** Function for score vector comparison

corresponding to the number of indifference relations (i.e., $r + 1$), because this is considered the least important one. If the score vectors are identical, the function returns 0 (line 12).

In contrast with the function *compareServices* presented in Fig. 4, the function *compareScores* induces a total order on the set of web service alternatives, thus allowing us to rank them accordingly. Using this algorithm, the 5 web service alternatives considered in our example will be ranked in the following order:

$(WS_4, WS_1, WS_2, WS_5, WS_3)$,

with $WS_4$ being the best alternative.

## 5 Conclusions and Future Work

In this paper we have introduced a new approach of ranking service alternatives based on the users QoS expectations. The users can define their requirements and preferences by using a simple and intuitive specification language that attaches conditions to lexicographic rules. Our approach facilitates the elicitation of preferences from clients, because it resembles the way people express trade-offs when reasoning about their preferences. The proposed method can deal with intransitive preferences and with situations where people are not fully aware of their preferences.

Our current efforts are directed toward designing and implementing a framework for dynamic web service selection that supports the handling of QoS preferences based on the approach presented in this paper. A prototype implementation of the ranking engine is available at `http://qospref.sourceforge.net/` and we plan to also offer open source implementations for the other components of our framework.

## References

1. Medjahed, B., Bouguettaya, A.: Service Composition for the Semantic Web. Springer (2011)
2. Maximilien, E.M., Singh, M.P.: A Framework and Ontology for Dynamic Web Services Selection. IEEE Internet Computing 8(5), 84–93 (2004)
3. Zeng, L., Benatallah, B.: QoS-Aware Middleware for Web Services Composition. IEEE Transactions on Software Engineering 30(5), 311–327 (2004)
4. Zhou, C., Chia, L.T., Lee, B.S.: DAML-QoS Ontology for Web Services. In: International Conference on Web Services, pp. 472–479 (2004)
5. Papaioannou, I.V., Tsesmetzis, D.T., Roussaki, I.G., Anagnostou, M.E.: A QoS Ontology Language for Web-Services. Advanced Information Networking and Applications, 101–106 (2006)
6. Zeng, L., Lei, H., Chang, H.: Monitoring the QoS for Web Services. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 132–144. Springer, Heidelberg (2007)
7. Day, J., Deters, R.: Selecting the best web service. In: Conference of the Centre for Advanced Studies on Collaborative Research, pp. 293–307 (2004)
8. Brafman, R.I., Domshlak, C.: Preference Handling — An Introductory Tutorial. AI Magazine, 58–86 (2009)
9. Marler, R.T., Arora, J.S.: The weighted sum method for multi-objective optimization: new insights. Structural and Multidisciplinary Optimization 41(6), 853–862 (2010)
10. Tversky, A.: Intransitivity of Preferences. Psychological Review 76(1), 31–48 (1969)
11. Manzini, P., Mariotti, M.: Choice by lexicographic semiorders. Theoretical Economics 7(1) (2010)
12. Herssens, C., Jureta, I.J., Faulkner, S.: Capturing and Using QoS Relationships to Improve Service Selection. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 312–327. Springer, Heidelberg (2008)
13. Chakhar, S., Haddad, S., Mokdad, L., Mousseau, V.: Multicriteria Evaluation-Based Conceptual Framework for Composite Web Service Selection. Evaluation and Decision Models: Real Case Studies. Springer, Berlin (2011)