U.P.B. Sci. Bull., Series C, Vol. 77, Iss. 3, 2015 ISSN 2286-3540

AN END TO END WEB SERVICE COMPOSITION BASED ON QoS PREFERENCES

Raluca IORDACHE[1], Florica MOLDOVEANU[2]

The service-oriented environment offers the opportunity to create new applications, by combining existing applications offered as services, in order to react to the business' increasing pressure of quickly delivering new applications. With a large number of web services offering the same functionality, choosing the web services that meet best the client's quality of service (QoS) requirements is a very important task. We introduce a QoS-aware end to end web service composition approach that handles all the stages from the web service discovery step, to the actual binding of the services. This approach uses our method of expressing non-functional preferences, which requires minimal effort on the part of the clients, but offers great flexibility in managing trade-offs. We define a QoS preferences ontology and use it in our semantic web service selection to choose an initial candidate list of services for every task in the orchestration model. Then, one concrete service is chosen from each candidate list and is boand to the corresponding task. This step involves computing and comparing the aggregated QoS of the resulting composite services. The selection is performed using a genetic algorithm.

Keywords: semantic web, QoS ontology, multidimensional QoS preferences, service binding, service composition.

1. Introduction

In SOA environments, loosely coupled services can be orchestrated into composite services in order to implement complex business processes. A composite service is usually described by an orchestration model involving a series of tasks. In order to execute a composite service, each task must be associated with a concrete web service that offers the required functionality. Moreover, the component services should be chosen in a way that assures the best quality of service (QoS) for the resulting composite service.

Web services have a dynamic nature: at every moment, a service may cease to exist or a new one may become available. Non-functional characteristics, such as the QoS, are also subject to frequent changes. Therefore, much research is

[1] PhD Student, "Automatic Control and Computers" Doctoral School, University POLITEHNICA of Bucharest, Romania, e-mail: riordache@outlook.com

[2] Prof., Dept. of Computer Science and Engineering, Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, e-mail: florica.moldoveanu@cs.pub.ro

directed toward automated, dynamic web service composition approaches. In this paper, we propose an end-to-end approach to this problem, which involves two main phases: discovery and binding.

The discovery phase is based on the using of ontologies in order to provide for each task a list of candidate services offering its required functionality. In the binding phase, for each task in the orchestration model, a unique service is selected from its list of candidate services. The selection process adopts a global optimization strategy devised to produce the composite service that best matches the aggregated QoS preferences expressed by the client. A novelty of our approach is that QoS preferences are also taken into account during the discovery phase, with the goal to restrict the number of elements in the list of candidate services for each task. This is achieved by extending the ontology with QoS preference information and using a local optimization strategy during the discovery phase.

Another important contribution of this work is the use of QoSPref [1], our preference specification method of expressing non-functional preferences, which offers great flexibility in managing trade-offs, but is at the same time very intuitive. This method leads to a simple algorithm for selecting web services, which does not require sophisticated multicriteria decision techniques.

We extend the OWL-Q [2] ontology to capture trade-off preferences expressed using our QoSPref notation. This ontology extension [3] allows a symmetric specification of the QoS properties of the request and the offer.

An initial list of candidate services is obtained for each task in the orchestration model by performing a semantic search based on the functional and non-functional requirements. For choosing the one concrete web service that will bind to a task from our composition, we compare them based on the overall QoS of the resulting composite services. This raises the issue of computing the aggregated QoS of a composite service based on the QoS of its component services. Most solutions to this problem are limited to composition models that can be represented as well-structured workflows. Our binding-as-a-service (BaaS) [4] implementation uses the aggregation method proposed by Yang et al. [5], which overcomes this restriction. The best mapping of concrete services to the tasks involved in the composition is foand using a genetic algorithm [6] that evaluates the fitness of a solution based on its aggregated QoS and on the QoS preferences expressed by the client using our QoSPref specification method.

2. QoS - Quality of Service

In SOA environments, the existence of numerous web services offering the same functionality needed by 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 fandamental role in the service selection. Non-functional characteristics of a web service, such as availability, cost, response time, or supported security protocols define the Quality of Service (QoS) concept.

Although considerable research has been done in the recent years, there is 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 for providing a complete solution. These issues include the design of suitable frameworks and architectures [7][8], which should provide ontologies for the formal specification of QoS metrics [9] [10], methods of obtaining current metric values [11] and web service selection algorithms based on user-specified QoS criteria [12].

3. The aggregated QoS of composite services

Various solutions have been proposed for the problem of estimating the aggregated QoS of a composite service, but they differ in the restrictions they impose on the topology of the composition. Most of them are limited to orchestration models that can be represented as well-structured workflows. Yang et al. [3] have introduced a method that overcomes these restrictions. This method, which is used in our binding-as-a service (BaaS) implementation, is presented in the remaining of this section. The input of this method is an orchestration model together with a binding that maps tasks to component services. An orchestration model is a directed graph with execution probabilities attached to its edges. The orchestration models are decomposed into orchestration components, which are subgraphs with a single-entry and single-exit point. The QoS is computed in a bottom-up manner for each orchestration component. Well-structured orchestration models, that is, models where each split gateway has a corresponding join gateway, are straightforward to analyze. Different aggregation formulas are provided depending on the type of the QoS attribute, which can be classified into three categories: critical path, additive and multiplicative. A preliminary step of the QoS aggregation method is to use the block-structuring technique introduced in [5] to transform an unstructured orchestration model into a maximally structured orchestration model.

We give an illustrative example that will be used throughout this paper in order to expose some of the issues related to the QoS-aware dynamic web service composition. We consider an online trading system offering services for trading various financial instruments. One of these services allows customers to buy both domestic and foreign stocks. The model can be transformed to a behaviorally equivalent model that is well-structured. The components that are irreducible using this technique are called rigid components and they are of two types: irreducible Directed Acyclic Graphs (DAG) and irreducible multiple-entry, multiple-exit (MEME) loops. The authors of [5] provide an algorithm that transforms irreducible DAG components in equivalent choice components. Irreducible MEME loops can be transformed using the block-structuring technique into equivalent rigid components where the concurrency is fully encapsulated within child components. For these equivalent components, the expected number of times that a node in the MEME loop is visited can be calculated using standard methods. This allows computing the QoS of the irreducible component by applying the aggregation formulas characteristic to each category of QoS parameters.

4. The problem of expressing preferences

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.

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 foand in various areas where optimal decisions involve trade-offs 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 [13] conclude that the weighted sum method "is fandamentally 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. This is based on Tversky's lexicographic semiorder notion where 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 [14] [15]. 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.

5. QoSPref - 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 the method based on our online trading system example introduced in [4]. Some of the tasks in this model (such as those for order registration) represent internal actions of the online trading system. Other tasks (such as those for getting stock quotes) require interaction with external systems. The online trading system implements the stock buying service as a composite web service. For the tasks involving interaction with external systems, it is necessary to find providers offering the required functionality as a web service. Usually, there are several alternatives for each of these tasks. For example, there are many web services that provide stock quotes. The online trading system has to decide which of the possible service components to bind to each task in its composition model. This service binding is a dynamic process, because over time, some component services may cease to exist and new ones may become available.

In order to be able to dynamically bind component services to the tasks specified in the composition model of the stock buying service, the online trading system must have an automated method of comparing composite services based on their QoS.

In our example, we consider that only the following QoS attributes are interesting for the online trading system: execution time, cost, and reliability. The QoS of the composite stock buying service is determined by the QoS of its service components, which are assumed to be known. However, it is not clear how to estimate the QoS of the composite service. While the aggregated cost can be easily computed by adding the costs of all component services, there is no obvious method for estimating execution time and reliability.

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 below:

```
constraints {
        cost < 1000,
         reliability > 0.95,
        executionTime < 60
}
```

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:

```
preferences {
        cost: low,
        reliability: high,
        execTime: low
 }
```

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*.

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. We add a few more details about the online trading system to illustrate why comparing composite services characterized by multiple QoS attributes is not a trivial task.

The executives of this system try to maximize their profit, therefore they see the cost as the most important QoS parameter. However, they are willing to ignore small cost differences (not exceeding 10 cents) if the composite service with a higher cost has better values for reliability and execution time.

For the customers of this system, it is very important that trading orders are executed as soon as possible. Therefore, the online trading systems guarantees that the execution time of its stock buying service does not exceed 30 seconds. For every violation of this agreement, the owners of the online trading system must pay a penalty proportional with the delay. This means that, when comparing two composite services, the execution time becomes the most important parameter if at least one of the compared services has an execution time exceeding the 30 seconds limit. It is clear that traditional methods such as weighted sum or parameter ranking are not appropriate for this scenario.

We consider that the executives of the online trading see the cost as the most important QoS attribute, followed by reliability and then by execution time. As mentioned before, the executives are willing to ignore small cost differences (not exceeding 10 cents) if the composite service with a higher cost has better values for reeliability and execution time. Furthermore, there are penalties to be paid if the execution time of the composite service exceeds 30 seconds. Therefore, when comparing two composite services, the execution time becomes the most important parameter if at least one of the compared services has an execution time exceeding the 30 seconds limit. 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(service1) OR condition(service2)
EXACTLY_ONE(condition)	condition(service1) XOR condition(service2)
ALL(condition)	condition(service1) AND condition(service2)
DIFF(attribute)	|service1.attribute - service2.attribute|

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.

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 Figure 1. (The preference rule indexes appearing at the left side of the figure are only informative and are not part of the preference specification.) The preferences corresponding to the above described scenario can be articulated as follows in our specification language:

```
        preferences {
1)              [AT LEAST ONE(execTime > 30)] execTime : low,
2)              [DIFF(cost) > 10] cost : low,
3)              reliability : high,
4)              execTime : low,
5)              cost : low,
        }
```

Figure 1. A more elaborate specification of preferences

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. Additionally, we introduce the notation $s_1 \succ_k s_2$ 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 $\prec_k$, defined by the following relations: $s_1 \prec s_2$, iff $s_2 \succ s_1$ and $s_1 \prec_k s_2$, iff $s_2 \succ_k s_1$

Our algorithm for comparing two web services based on the preferences expressed using our conditional lexicographic approach examines all entries in the preferences block in the order in which they appear. If the current preference rule has no attached condition, or the attached condition evaluates to true, the values corresponding to the attribute specified by this entry are compared. If the attribute values are not equal, 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. Otherwise, the execution continues with the next preference rule. The algorithm returns either an indifference relation between the two web services, or a tuple that identifies a relation of type $\prec_k$ or $\succ_k$ between them.

In a series of experiments, Tversky [6] 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. An illustrative example of how our method is used in a context containing intrasnsitive preferences is given in our paper [1].

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 score vector of one web service contains for every preference rule the number of times this web service was prefered to another one due to this specific preference rule and a last entry containing the number of times this web service was indifferent to another one.

Using the score vectors, we are able to provide an algorithm for the ranking of web service alternatives. Please refer to [1] for pseudocode and further details. This algorithm induces a total order on the set of web service alternatives, thus allowing us to rank them accordingly, eliminating the intransitivity.

6. Ontology extension for expressing tradeoff preferences

Semantic web has gained popularity in the recent years, in part due to the need to automate the service discovery process. The use of ontologies facilitates machine reasoning, allowing the implementation of complex web service selection engines that offer accurate results. The selection of web services, as well as the specification of the process workflow based on ontologies has been also advocated by the authors of [16]. Usually, the web service selection process includes the discovery step and the selection step. The discovery step involves searching for appropriate web services, based on functional criteria. During the selection step, the best fitting service is selected from the services discovered in the previous step, by taking into account the client's non-functional requirements. Therefore, ontologies for semantic web should be able to deal with both functional and non-functional requirements.

We argue that the capability to express QoS preferences and trade-offs between them is crucial for selecting the best 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.

Currently, there is no established standard for describing the QoS properties. After analyzing the existing QoS ontologies we have chosen to extend the OWL-Q ontology, a complex ontology designed into several facets that can be extended and enriched independently. This ontology allows a symmetric specification of the QoS properties of the request and the offer. For the matchmaking algorithm, the approach transforms the comparison to a Constraint

Satisfaction Problem and extends the existing CSP-based approaches. OWL-Q addresses the problem of specifying the requester's priority for QoS constraints, by allowing the requester to provide weights to metrics of his interest. The weights express the impact of the attributes and allow the ranking of the offers.

We argue that providing weights for the metrics isn't enough for capturing complex QoS requirements. In order to use the flexibility of our QoSPref method for expressing trade-off preferences in a semantic context, we extend the OWL-Q ontology to use our conditional lexicographic method.

In OWL-Q [2] the QoS offers and requests are defined by the QoSSpec Facet. The QoSSpec class contains the QoS description of a web service. It contains several attributes like the cost of using the service and the currency, security and transaction protocols, the validity of the offer. QoSSpec is separated into two subclasses, disjoint to each other: QoSOffer and QoSDemand where the WS providers and requesters can define in the same way, symmetrically, their QoS constraints.

The WS requester can provide constraints by using the QoSDemand class and can also provide weights to metrics of his interest, by using the QoSSelection class. The QoSSelection class contains list of <metric,weight> entries. The weight value can have the value of 2.0 if it is a hard constraint or a value in (0:0; 1:0) if it is soft.The entry <metric,weight> is defined by the QosSelectElem class and the QoSSelectElemeList contains a list of QosSelectElem-s.

OWL-Q also contains a Metric Value Type Facet that describes the types of values a QoS metric can take. The MetricValueType class has two subclasses indicating the direction of values for the QoS metric that owns one of these two subtypes: PositivelyMonotonic value types have a direction of values from the lowest to the highest value where the highest value is mapped to the highest quality level that can be achieved while a NegativelyMonotonic value type has an opposite direction of values where the highest quality level is mapped to the lowest value. As an example, a QoS metric measuring the Availability QoS attribute has as value type a positively monotonic value type while a ReponseTime QoS Attribute has a negatively monotonic value type, as the lowest values are the better ones. Thus, when describing a preference rule from out QosPref approach, we don't need to explicitly write the direction flag of a metric anymore, as this is already described by the metric itself.

For our extension of the OWL-Q Requirements Specification to capture preference trade-offs, we can keep from OWL-Q the QoSOffer description, as the provider specification remains unmodified. For this reason, we choose to extend the existing facet by adding new classes for the preference selection, and not create another facet.

We define a new QoSSelectionWithTradeoffs class that will incorporate a list of preferences, described similar to our QosPref's notation. The hard constraints are specified using the QoSDemand class, just like in the OWL-Q ontology. A preference is described by a QoSPreference class that incorporates a preference rule. As already seen, a preference rule has three components: an optional condition, an attribute indicating the QoS metric used in comparisons and a direction flag stating which values should be considered better. As the direction flag is specified using the MetricValueType class from OWL-Q, a preference rule has to include the optional condition and the metric. A preference rule entry looks like that: <(optional condition) metric>.

For the optional condition the three logical operators AT_LEAST_ONE (OR operator), EXACTLY_ONE (XOR operator), ALL (AND operator) and the arithmetic operator DIFF (-) as seen in *Table 1* have to be included in the ontology. The optional condition can be described as <operator, metric>.

Another observation is that the order of the preferences is an important factor for the ranking algorithm, with the first preference being the most important. So, the order of preferences in the preferences list is important. To make the semantic notation easy to follow we choose to explicitly attach the priority to every preference rule so that an entry in the QoSPreference class will look like that: <priority, (condition) metric>. The QoSRequest class will point to both QoSSelectionWithTradeoffs and QoSSelection class. The requester can decide which QoS specification he wants to use, depending on the use case. The ontology extension allows the requester to use a notation to flexibility define complex constraints and trade-offs of preferences in a semantic context.

7. BaaS -Binding as a Service

There are several web service composition frameworks, with different architectures and methodologies. Nonetheless, service binding is a task required by all these frameworks. Therefore, it is useful to offer this functionality as a service. The typical client of a BaaS provider is a module of a web service composition framework, which needs to find the best mapping of concrete web services to the tasks of a composition model.

We provide a QoS-aware BaaS implementation [4] based on the QoS aggregation method of Yang et al. [5] and on our preference handling approach detailed in the previous section.

A web service request sent to our BaaS provider must contain the following information: the orchestration model, the list of QoS attributes, for each task in the orchestration model, a list of concrete web services offering the required functionality, the QoS constraints and the QoS preferences.

The orchestration model is represented as a workflow with execution probabilities attached to its edges. If probabilities are missing, our implementation will assign default probabilities. Edges starting from an XOR gateway are assigned a probability of *1/k*, where k is the number of outgoing edges of the given XOR gateway. All other edges are assigned a probability of 1.

The list of QoS attributes must contain information about the aggregation category of each attribute. The list of concrete web services offering the required functionality of a given task must specify for each concrete web service its QoS values. Not all web services have all QoS attributes of the composite service. For example, a composite service that converts data sets to graphic charts may have a QoS attribute indicating the number of colors of the resulting image. The composite service may have a component service that sorts the data set. The number of colors is clearly not a QoS attribute of the sorting service. In situations where a QoS attribute is missing for a component service, our implementation provides default values, in accordance with the aggregation category of the missing QoS attribute.

Some of the tasks in an orchestration model may be internal actions. For these tasks, the list of concrete web services implementing their functionality is empty.

The QoS preferences are specified using the preference notation introduced in the previous section. Optimizing the aggregated QoS of a composite service is an NP-hard problem. An exhaustive search is feasible only for simple compositions models, having a small number of tasks and a small number of available services for each task. To overcome these issues, the BaaS implementation uses a genetic algorithm, which finds the best mapping of concrete services to the tasks involved in the composition. The algorithm uses the method presented above, in the QoSPref Section, to estimate the fitness of the mappings that make up a population of candidate solutions.

A genetic algorithm maintains a population of chromosomes, where each chromosome encodes a possible solution of the problem. In our case, a chromosome encodes a possible mapping of web services to tasks. The chromosome is structured as a vector of n elements, where n is the number of tasks in the orchestration model. The value of the element i in this vector is an integer indicating the index of the component service assigned to the task i. Our genetic algorithm uses the two-point crossover operator for recombination. Mutations are performed by randomly choosing a task and randomly changing the index of its assigned component service.

A peculiarity of our approach for preference specification is that the fitness of a solution can only be evaluated in the context of a given population, because the ranking algorithm performs pair wise comparisons of all candidate solutions. Therefore, it is not possible to offer an absolute value for the fitness of a solution. Our genetic algorithm computes the fitness of a solution based on its ranking in the current population, by assigning the maximum value to the top ranking solution and the minimum value to the solution at the last position in the ranking.

Since there is no absolute value for the fitness of a solution, checking the occurrence of the third condition is not a trivial operation. In order to solve this issue, the genetic algorithm maintains a list of best-so-far solutions. At the end of each generation, the best solution in the current population is searched in the list of best-so-far solutions. If not already present, it is added to this list and ranked against the other elements. An improvement has occurred only if the current best solution is at the top of the resulting ranking. The size of the list of best-so-far solutions is limited by a value configured as a parameter of the algorithm. If, as a consequence of adding the current best solution to the list of best-so-far solutions, its size exceeds the limit, the element with worst ranking will be removed.

For a detailed description of the genetic algorithm, usage examples and experimental results please refer to our paper [6].

8. Conclusions

We propose a dynamic web service composition approach that can deal with complex QoS preferences. We offer an end to end solution, starting from the service discovery step, based on a preferences enriched semantic search, to the actually service binding step. In our work, we have combined powerful technologies. We use a semantic search that takes into consideration the required QoS preferences in order to restrict the number of elements in the list of candidate services for each task. This is achieved by using our ontology extension to capture complex QoS preference tradeoffs. We introduce a new approach of ranking service alternatives based on the user's QoS expectations. The users can define their requirements and their preferences by using a simple and intuitive specification language. The service alternatives are compared using a simple algorithm that allows dealing with tradeoffs of the preferences.

To overcome the challenges brought by measuring the QoS of a composition of web services we use the aggregation method of Yang et al. [5], which has the major advantage of being able to deal with unstructured orchestration models. Finally, we use a genetic algorithm for finding the best mapping of component services to the tasks involved in a service composition.

We offer a extension for the OWL-Q ontology, implemented with Protégé. For the (BaaS) provider we offer a prototype implementation written in Java as an open source project at: http://baas.sourceforge.net/. A prototype implementation of the ranking engine based on our method of preferences specification, offering a graphical interface as well, is available at http://qospref.sourceforge.net/.

R E F E R E N C E S

[1] *R. Iordache and F. Moldoveanu*, „A conditional lexicographic approach for the elicitation of QoS Preferences," in s Lecture Notes in Computer Science vol. 7565, pp 182-193,

Rome, 2012.

[2] *K. Kritikos and D. Plexousakis*, „OWL-Q for Semantic QoS-based Web Service Description and Discovery," in s Fifth IEEE European Conference on Web Services, Halle, Germany, 2007.

[3] *R. Iordache and F. Moldoveanu*, „QoS-aware web service semantic selection based on preferences," in s 24th DAAAM International Symposium on Intelligent Manufacturing and Automation, Zadar, Croatia, 2013.

[4] *R. Iordache and F. Moldoveanu*, „A web service composition approach based on QoS preferences," in s Proceedings of the 6th IEEE International Conference on Service Oriented Computing & Applications (SOCA 2013), Kauai, Hawaii, 2013.

[5] *Y. Yang, M. Dumas, L. García-Bañuelos, A. Polyvyanyy and L. Zhang*, „Generalized aggregate Quality of Service computation for composite services," Journal of Systems and Software, Nr. 85(8), pp. 1818-1830, 2012.

[6] *R. Iordache and F. Moldoveanu*, „A genetic algorithm for automated service binding," in s 24th DAAAM International Symposium on Intelligent Manufacturing and Automation, Zadar, Croatia, 2013.

[7] *E. M. Maximilien and M. P. Singh*, „A Framework and Ontology for Dynamic Web Services Selection," IEEE Internet Computing, Bd. Bd. 8, Nr. 5, pp. 84--93, 2004.

[8] *L. Zeng and B. Benatallah*, „QoS-Aware Middleware for Web Services Composition," IEEE Transactions on Software Engineering, Issue 5, Bd. Bd. 30, Nr. 5, pp. 311--327, 2004.

[9] *C. Zhou, L.-t. Chia and B.-s. Lee*, „DAML-QoS Ontology for Web Services," in s Proceedings of IEEE International Conference on Web Services, San Diego, CA, USA, 2004.

[10] *I. V. Papaioannou, D. T. Tsesmetzis, I. G. Roussaki and M. E. Anagnostou*, „QoS Ontology Language for Web-Services," Vienna, Austria, 2006.

[11] *L. Zeng, H. Lei and H. Chang*, „Monitoring the QoS for Web Services," in s International Conference on Service Oriented Computing, 2007.

[12] *J. Day and R. Deters*, „Selecting the best web service," in s Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research, 2004.

[13] *R. T. Marler and J. S. Arora*, „The weighted sum method for multi-objective optimization: new insights," Structural and Multidisciplinary Optimization, Bd. 41, Nr. 6, pp. 853-862, 2010.

[14] *A. Tversky*, „Intransitivity of Preferences," Psychological Review, Bd. Vol. 76, Nr. Jan 1969, pp. 31-48, 1969.

[15] *P. Manzini and M. Mariotti*, „Choice by lexicographic semiorders," Theoretical Economics , pp. 1-23, 2012.

[16] *A. Costan and V. Cristea*, „A workflow management engine for scientific applications," in s U.P.B. Sci. Bull., Series C, Vol. 73, Iss. 2, Bucharest, Polytechnic Institute of Bucharest, 2011, pp. 73-88.