Translating User Preferences into Fuzzy Rules for the Automatic Selection of Services

Ioana Şora, Doru Todinca, Cătălin Avram
Department of Computers
Politehnica University of Timisoara, Romania

***Abstract*—This article proposes an approach for including user preferences and quality of service characteristics in the selection process of services. Our approach consists of a Domain ontology for the service description vocabulary, and a trader that facilitates user-preference-based service selection, combining imperfect service matching and ranking algorithms. The novelty of our approach lies in the fact that we automatically generate fuzzy rules starting from individual user preferences and use them in a fuzzy inference process that ranks the service candidates. We present our experiments to evaluate our approach using a prototype implementation of a service broker.**

I. Introduction

Service Oriented Architectures are based on the notion that programs can be constructed by composing independent services. A service is defined as "a loosely-coupled, reusable software component that encapsulates discrete functionality which may be distributed and programmatically accessed" [1], [2]. The standard approach in Service Oriented Computing is that Service Providers host network-accessible software modules (implementation of given services) and define service descriptions that are published in Service Registries. Clients or Service Requesters use the published service descriptions in order to find services appropriate to their needs. After finding the service, the requester uses the service description to bind to the provider and invoke the service. A service can be both a provider and in the same time a client for other services. Service compositions can act as services as well and be published as such.

In the field of service discovery and selection, the major research challenges include enhancing service discovery and selection with semantic and non-functional aspects. [3]. The main challenge of service discovery is to use tools to automatically (with minimum user involvement) and accurately discover and select services. Achieving automated service discovery and selection requires adding semantic annotations and descriptions of QoS characteristics to service definitions on one side, and using a formal request language able to precisely describe the requester's needs or desired services on the other side.

Current practices have defined standards for Service Registries like UDDI (the Universal Description Discovery and Integration specification) that defines how to publish and discover information about Web services. For the description of web services, the current standard is WSDL, the Web Services Description Language [4]. WSDL can be used to describe functional aspects of services, their ports and messages.

An important research effort is directed toward semantic matchmaking, that will permit to ensure that functionalities defined in similar syntactics have similar meanings. Description languages like WSDL-S and OWL-S come in here, making possible the description of semantic concepts.

Another important research track investigates ways to include user preferences and quality of service characteristics in the selection process. These factors should have the final word in the selection process if multiple Web services provide the same functionality, but they could also be a decision factor in the choice between several alternative functionalities. Our work proposes an approach for dealing with these issues.

We have developed a prototype implementation of a service broker, and we use fuzzy logic in order to deal with the issue of selecting the optimal match for each request, according to its individual QoS preferences, from a set of imperfect candidates.

Our approach starts by proposing a way to complete the QoS information in domain ontologies with fuzzification categories, as we describe it in Section II. In Section III we present how users can describe their requests and preferences in our service broker tool. Section IV introduces our approach of automatically generating a set of fuzzy rules for each set of individual preferences. The ranking of candidates is done by testing every possible candidate found in the service repository against this set of generated fuzzy rules. Some experimental results are presented in Section V. Section VI discusses our approach in the context of related work.

II. Domain Ontology

In order to enable automatic matching while keeping a low degree of formalism in the description of services, most approaches rely on the concept of domain ontologies [5] [6]. An ontology is an explicit formal specification of how to describe the concepts that exist in a specific domain and the relationships between them. Of course, for its practical success, an ontology must be widely agreed upon, accepted and promoted by standardizing or professional organizations.

In its simplest form, a domain ontology would specify the valid vocabulary of describing (naming) functional and non-functional properties that are allowed to occur in service descriptions (examples: availability, reliability, version stability, response time, cost, payment method, concurrency capacity, publisher reputation, etc.)

In order to increase its usability, a good domain ontology has to be more than a simple taxonomy of domain vocabulary.

Several approaches of domain ontologies have been proposed, from simple ones that define simple name-value pairs [7] to more complex ones [6],[8], [9].

None of these approaches is fully appropriate for our goal of facilitating imprecise (fuzzy) matching, where we need a domain ontology that can help in defining categories through linguistic variables. For example, the response time could be described with the terms very fast, fast, slow, very slow. If the domain of the variable is known, then a domain expert can trace the membership functions for the linguistic terms. In our work, we use triangular/trapezoidal shapes of membership functions. For example, in order to characterize the response time of a *TravelScheduler* service, the terms can be described like in Figure 1.

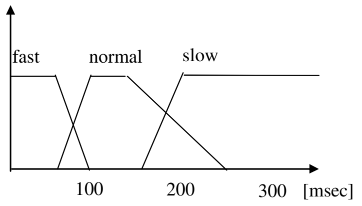


Fig. 1. Membership functions for fuzzy terms defining the response time of a *TravelScheduler* service

These terms, however, in their number, shape and margin values, are valid only for the class of services of type *TravelScheduler*. For another type of service, for example *WeatherForecast*, the response time expectations are different and the membership functions for the response time look very different, like in Figure 2. This is because the expected response times vary depending on the type of the requested service: while 200 seconds is considered a normal amount of time to receive a personalized travel schedule, the same amount of time is considered to be a bad performance (very slow) if it is the response time of a service broadcasting weather forecasts.

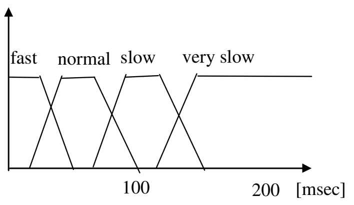


Fig. 2. Membership functions for fuzzy terms defining the response time of a *WeatherForecast* service

Thus, we consider that it is very important that the domain ontology distinguishes between general valuable non-functional properties (i.e., availability) and context-dependent non-functional properties, that can be defined only in the context of specific functional properties (i.e. response time, cost). We also consider that non-functional properties can be either of a value-type (as in the case of availability, average response time or other measurable quantities), or of a description-type, (as in the case of publisher-reputation, that cannot be described by measured quantities but only by descriptions like excellent, good, fair, poor).

For each non-functional property (general valuable or context-dependent), following description items have to be provided:

- domain: the range of values for measurable quantities, or a default range 0-100 for description-only properties.
- optimization direction: specifies what is better, for the current property, a bigger value (as for availability) or a smaller value (as for response time)
- fuzzy terms: the fuzzy terms which can be used to describe this property have to be enumerated, for each term providing its name and the shape of its membership function as defined by a domain expert.

We do not include in our ontology prototype aspects related to metrics and measurement units (for example, if clients requests and provider descriptions use different time units - minutes vs seconds, then appropriate conversions should be done). These are treated in complex approaches of ontologies like QoSOnt [6], [8] and are definitely needed in a real-life implementation, but in this work where we keep the focus on the imprecise matching issues they are not relevant for our main purpose.

Service descriptions must be augmented with a description of their non-functional properties. This can be done either by their publisher (for example, the publisher may specify the average response time and the cost of the service) or by specialized external monitoring or rating agents (for example, the availability or reputation). Independent of the source, the non-functional description must use ontology-terms in order to specify the service properties. Properties can be specified in service descriptions either as crisp values, or as fuzzy terms. It is not mandatory for a service description to specify all possible non-functional properties classified in the ontology for its corresponding functionality, but in the evaluation process the unspecified properties are considered to be in the worst fuzzy category from their domain ontology.

Our implementation uses an XML schema for the non-functional properties description, and examples of service descriptions in a repository is shown in Figure 3.

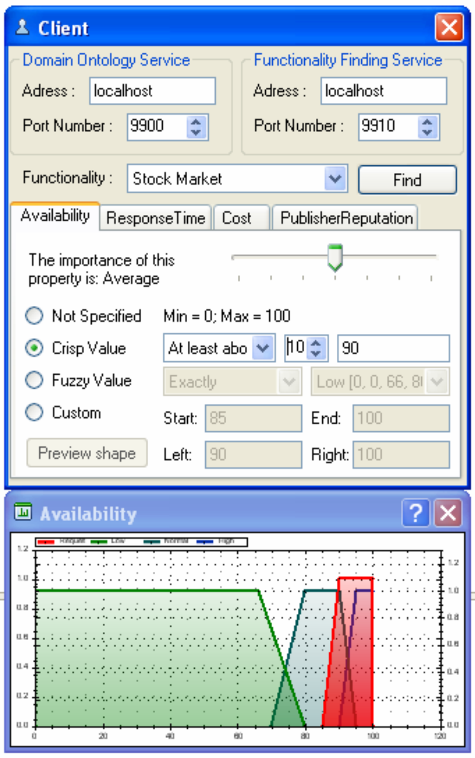
The examples in Figure 3 show four service descriptions, two implementing the functionality of a *TravelScheduler* and two implementing the functionality of a *WeatherForecast*. All service descriptions specify the response time property. For services S0001 and S0002 (of type *TravelScheduler*) their response time is categorized according to the fuzzy terms described in Figure 1, while for services S0003 and S0004 (of type *WeatherForecast*) the response time is categorized according to the fuzzy terms described in Figure 2.

III. Specification of Client requirements

The Client requirement has to specify both functional and non-functional requirements. The functional requirements are always considered non-negotiable. The non-functional requirements can be of several types: non-negotiable (exact value or

```
<service id="S0001" name="Happy Camper"
    functionality="TravelScheduler">
  <prop name="Availability" fuzzy="High" />
  <prop name="ResponseTime" crisp="500" />
  <prop name="PublReputation" fuzzy="Medium" />
  </service>
 <service id="S0002" name="De Luxe Tours"
    functionality="TravelScheduler">
  <prop name="Availability" crisp="90.7" />
  <prop name="ResponseTime" crisp="100" />
  <prop name="Cost" crisp="10" />
  </service>
<service id="S0003" name="LocalWeather"
    functionality="WeatherForecast">
  <prop name="Availability" crisp="80.8" />
  <prop name="ResponseTime" crisp="200" />
  <prop name="PublReputation" fuzzy="good" />
  </service>
 <service id="S0004" name="WeatherToday"
  functionality="WeatherForecast">
  <prop name="Availability" crisp="90.3" />
  <prop name="ResponseTime" fuzzy="Fast" />
  <prop name="PublisherReputation" fuzzy="bad" />
  <prop name="Cost" crisp="0" />
  </service>
```

Fig. 3. Example: service descriptions in a repository



Client

Domain Ontology Service
Adress : localhost
Port Number : 9900

Functionality Finding Service
Adress : localhost
Port Number : 9910

Functionality : Stock Market — Find

Availability | ResponseTime | Cost | PublisherReputation

The importance of this property is: Average

Not Specified — Min = 0; Max = 100
[✓ Crisp Value] — At least abo — 10 — 90
Fuzzy Value — Exactly — Low [0, 0, 66, 8
Custom — Start: 85 — End: 100
Preview shape — Left: 90 — Right: 100

Availability

Fig. 4. Example of specifying individual user preferences

sharp interval) or negotiable (around a value or around an interval) or best-possible.

The way the user may specify its requirements is the following:

- crisp value: the user specifies a crisp value (i.e, 90 for availability) or a crisp interval, and wether it is negotiable or not. If it is negotiable, it can be described in following terms: *about*, *at least about*, *at most about*. In this case, a fuzzification of the value is done automatically, with a slope that is percentually specified by the client.
- fuzzy value: the user specifies a fuzzy value, either a term from the domain ontology for the corresponding property (i.e., the user may request *medium* availability if he considers that a medium-effort solution is satisfactory for his request) or an advanced user can edit a customized form of a fuzzy value.

Figure 4 presents an example of how a client specifies its non-functional preferences for finding a service with *Stock-Market* functionality. For availability, the client specifies the condition *at least about 90*, with a fuzzification slope of 10%. This user defined shape is comprised of the term *high* in its entirety and a slice of the term *medium* referring to the availability property from the domain ontology.

IV. Rule Generation Strategies

The client requirements are automatically processed and transformed into fuzzy rules. The premises of the fuzzy rules specify possible combinations of values for the attributes, and the corresponding conclusion specifies the degree of acceptability for such a candidate service. The conclusion is the variable *SelectionDecision*, and has terms such as *StrongReject*, *WeakReject*, *WeakAccept*, *StrongAccept*, that cover the range [0,1]. The ranking of candidates is done by testing every possible candidate found in the service repository against the set of generated fuzzy rules, using a fuzzy inference machine. For each candidate, the value of *SelectionDecision* in conclusion gives its ranking score.

We investigated different strategies for the generation of these rules. First, we used an approach where attributes in client requirements were mapped to terms existing in the domain ontology, and second we developed an approach where each client requirement generates a specific neighborhood ontology.

A. Strategy A: based on mapping to existing domain ontology

This strategy is similar to the one in our previous work described in [10]. The request uses fuzzy terms from the domain ontology to specify the requested values. For example, property *Availability* is described in the domain ontology with the terms *Low*, *Medium*, *High*, *VeryHigh* and property *ResponseTime* is described in the domain ontology with the terms *VerySlow*, *Slow*, *Medium*, *Fast*, *VeryFast*. These are the terms used in the rules premises. A client that requires at least medium availability and fast response time will be translated into a set of rules like:

RA1: If Availability=Medium and ResponseTime=Fast then SelectionDecision = StrongAccept

All combinations containing better values of these attribute will generate rules with the same strength in conclusion, as they have an equal value for the user, for example:

RA2: If Availability=High and ResponseTime=Fast then SelectionDecision = StrongAccept

RA3: If Availability=VeryHigh and ResponseTime=VeryFast then SelectionDecision = StrongAccept

For the less good values, the strength of the conclusion will be diminished proportionally with the distance from the ideal situation. For a match with a direct neighbor, the conclusion strength goes one term down, for a match with a second neighbor the conclusion strength goes two terms down, etc.

RA4: If Availability=Low and ResponseTime=Fast then SelectionDecision= WeakAccept

RA5: If Availability=Low and ResponseTime=Medium then SelectionDecision= WeakReject

RA6: If Availability=VeryLow and ResponseTime=Fast then SelectionDecision=WeakReject

The above listed rules are only a few examples from the complete set. The automatic rule generation system produces, in a similar way, the complete set of rules for all possible combinations of terms in premises and corresponding terms in conclusions. In this case, where we have two variables in the premise, with 4, respectively 5 terms, the total number of generated rules is 20.

B. Strategy B: based on a generated neighborhood ontology

Strategy A does not allow a request to differentiate between values that belong to the same categories in the domain ontology. Often it is the case that a request specifies more differentiated values (a range of values) for a property, for example a range of "at least about 75" or "at least about 85" for availability, both values being in the standard medium category. Thus we implemented a second strategy for the rule generation, that starts with creating a neighborhood ontology for the required value or range of values. A new neighborhood ontology is generated, describing for each property the meaning of exactly matching the target, or being near or far away from the target. The new linguistic variables in premises are *MatchingAvailability* and *MatchingResponseTime*. For each property, the limits of the given range of values are fuzzified (using a percentual approach for smoothing the limits) and this results in the *Exactly* term of the corresponding linguistic variable, and has a trapezoidal shape. Then, terms like *NearLeft*, *FarLeft*, *VeryFarLeft*, *NearRight*, *FarRight*, *VeryFarRight* are generated, with trapezoidal shapes automatically calculated considering a percentually distance from the center in *Exactly*. There is also the option that the client starts by specifying directly a custom shape for *Exactly*, and the other neighborhood terms are generated as described above.

The rules will have in their premises variables from the generated neighborhood ontology, instead of the fixed variables from the domain ontology. The strategy of rule generation is similar as described above for strategy A. The first rule corresponds to the situation when all properties match the fuzzy term of *Exactly*:

RB1: If Availability=Exactly and ResponseTime=Exactly then SelectionDecision= StrongAccept

For the less good values, the strength of the conclusion will be diminished proportionally with the distance from the ideal situation, similar with strategy A.

RB2: If Availability=NearLeft and ResponseTime=Exactly then SelectionDecision = WeakAccept

RB3: If Availability=NearLeft and ResponseTime= NearRight then SelectionDecision = WeakReject

The rule generation strategy can be configured with parameters like: number of terms in conclusions (how many degrees of acceptance are defined) and number of generated neighborhoods (how many categories there are to describe an inexact match for a property).

V. Results

We have studied the influence of the rule generation strategy over the overall quality of the ranking result. We consider the following criteria in order to appreciate the quality of a ranking strategy:

1) Ranking hierarchy of solutions: is it the same hierarchy as the ideal one or some inversions appear ?
2) The average ranking step (the distance between candidates ranked on consecutive positions): We prefer a ranking strategy that produces a clear hierarchy, with significant distances between the scores of consecutive ranked candidates
3) The range covered by the ranking scores: We consider that, for the same ordering of the candidates, it is desirable that their scores are spread over a wider range, as this simplifies the automatic defuzzification in order to take the final select/reject conclusion.
4) The number of distinct ranks: We consider that a good strategy differentiates as much as possible between all candidates, and does not repeatedly rank several candidates on similar scores

The tests described here refer to 26 candidates that had to be ranked. The user preferences can specify values for 5 properties (there are 5 linguistic variables in premises of rules). We compare strategies A and B for the generation of rules, as well as the influence of the number of fuzzy terms in conclusion. The number of terms for the variable *SelectionDecision* in the conclusion varied from 2 (*accept*, *reject*) to 8 (*very strong accept*, *strong accept*, *weak accept*, *weak reject*, *strong reject*, *very strong reject*). For the variables in the premises, the number of neighbors counted in the rule generation process was (up to) 2 neighbors on each side.

Figure 5 presents our results. The values are averages obtained from solving 3 different requests with different combinations of preferences.

From this Figure, it results that the solution Strategy B with a medium number of terms (4-5) for *SelectionDecision* is an optimal solution, from point of view of all the criteria 2-4: the average ranking step, the range of ranking scores and the number of distinct ranks.

Strategy A fails on the 4th criteria regarding the number of distinct ranks, it does not differentiate enough between candidates, as it ranks many of them on the same positions.

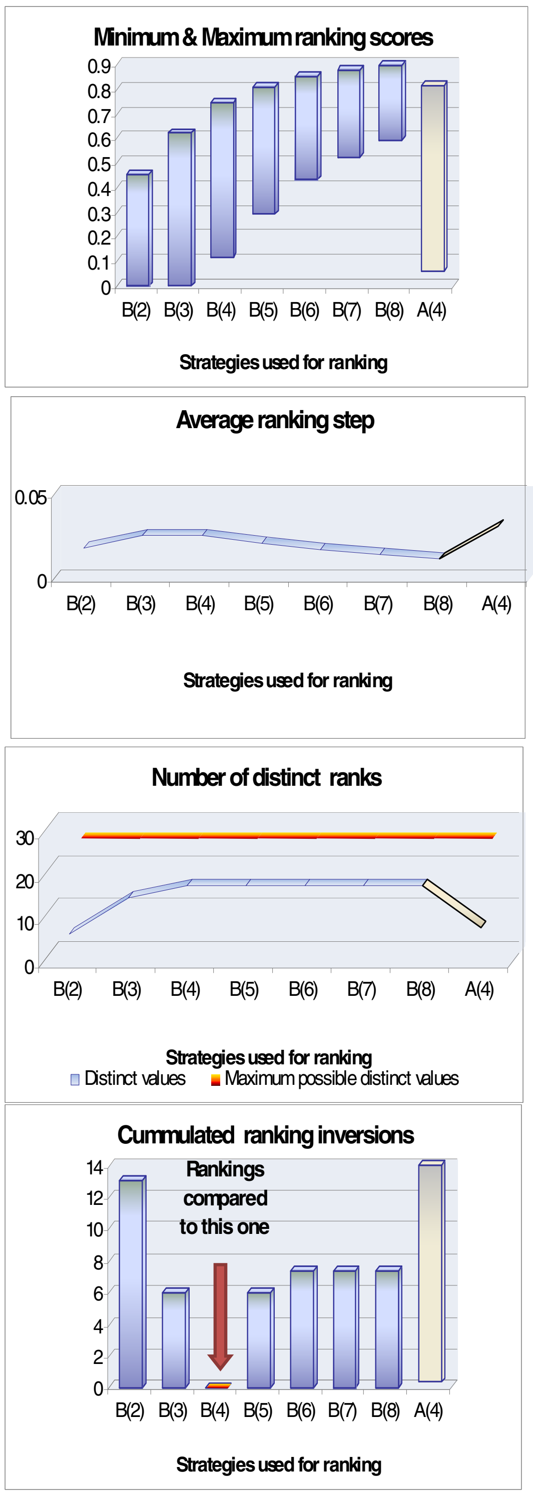


Fig. 5. Experimental comparison of ranking strategies

This was expected, as this strategy unifies user preferences in terms of the predefined categories in the domain ontology.

For small values (2-3) of the number of outcomes, strategy B establishes a relatively small number of distinct ranks, and tends to score too many candidates as unacceptable, the range of scores does not pass much over the final acceptance threshold of 0.5. For bigger values (6-8) of the number of outcomes, strategy B tends to classify too many candidates as acceptable.

Small inversions in rankings have been counted across the different strategies. As it was difficult to establish an independent absolute reference hierarchy, we took as a reference the hierarchy produced by strategy B in its optimal form B(4). We computed the cumulated ranking inversions for a pair of ranking hierarchies as the sum of position differences for all candidates.

VI. Related Work

QoS-aware selection of services is a hot topic in today's research. There are many approaches dealing with it, including approaches that rely on artificial intelligence.

Several works start with proposing models of QoS attributes and QoS ontologies. In [7], the authors propose an QoS model that includes generic and domain or business specific criteria, concentrating on criteria that can be collected and measured objectively. A more complex ontology is proposed in [5], which defines metrics and introduces the concept of derivation of different QoS parameters. Another hierarchic modeling approach is described in [11] - Analytic Hierarchy Process (AHP) serves as decision making model, the authors build a QoS Meta-model.

What we were missing in this domain of QoS modeling, was the possibility of defining and using categories for the different attributes. Our approach has the main goal to fill this gap in the domain of QoS modeling by permitting to define semantic categories (low, big, fast, slow, cheap, expensive, etc), further used as terms of fuzzy linguistic variables, for each QoS attribute and in the specific context of different functionalities.

For selection and ranking, different approaches are used: In [7], the authors use a normalized m*n Quality matrix to evaluate n candidate services for m criteria. [12] uses Singular Value Decomposition SVD, based on decomposing the Quality and Web Services matrix (similar as in MCDM) in order to identify groups of Services with similar qualities and rank them accordingly. [13] considers service selection based on maximizing a utility function under cost constraints. Utility functions reflect the importance of different attributes in a request. These approaches perform the ranking of services according to a specific user request.

Some approaches include also a form of feedback for updating measured QoS informations. In [14], the authors propose an adaptive framework to measure and update availability and rank services on availability. Other approaches use constraint programming to check QoS conformance [9], or hybrid approaches like [15], which combine Integer Programming, genetic algorithms, and case-based reasoning to tackle the QoS-aware service composition problem.

In the above mentioned approaches, trade-offs between attributes required by client and provided by candidate services are usually limited to a certain strategy for assigning scores to candidates, usually weighted by the relative importance of the attribute. Improvements in the selection algorithm tackle the problem of reducing the computational complexity of the NP-hard selection problem. Fuzzy logic is a solution for both the computational complexity and the matching with imprecise QoS constraints. An overview of fuzzy approaches is presented in [16].

Most of the fuzzy approaches are variants of Fuzzy Multi Criteria Decision Making ([17], [18], [19]) or a version of Fuzzy decision by a committee of evaluators [20]. These approaches are similar in a way that they can be assimilated with different types of fuzzy decision matrices. Also, each approach may use different types of fuzzy parameters (forms of fuzzy terms, inference strategies, fuzzification/defuzzification strategies), but no comparison is done among these.

A few approaches are based on Fuzzy Rules. The work described in [21], [22] uses fuzzy rules to express user preferences. Intuitively fuzzy rules express which combination of attributes is the user willing to accept to which degree. Attribute values and degrees of acceptance are fuzzy sets, but no specification of how these linguistic variables are mapped to crisp values is given. This approach resembles strategy A in our work. The user is limited to specify as values for its preferences the predefined terms that describe the attribute in the domain ontology. Our approach is more complete, defining the meaning of fuzzy terms in context of individual functionalities in the domain ontology. Also, in our approach additional rules are also generated (automatically and gradually diminishing acceptance degree when attributes match only some neighboring categories).

Besides this strategy, we also developed strategy B for rule generation, when fuzzy categories are automatically generated from a precise user requirement and also ranking rules are generated in order to describe the automatically and gradually diminishing acceptance degree. This strategy is a generalization of the FMCDM algorithms, and we intend to prove in our future work that it is more flexible and has a bigger and more controllable expressive power than FMCDM, especially in the case when several properties with different importance degrees are considered.

VII. Conclusion

In this work we use fuzzy logic for selecting the optimal match for each individual combination of QoS preferences, from a set of imperfect candidates. Our approach starts by proposing a way to complete the QoS information in domain ontologies with fuzzification categories. The novelty of our approach lies in using fuzzy inference for ranking the candidates, but based on sets of automatically generated fuzzy rules for each set of individual preferences. Fuzzy rules have a big expressive power, and the fact that they are generated automatically makes this approach user-friendly.

References

[1] J. Sommerville, *Software Engineering*, 8th ed. Addison Wesley, 2006.

[2] T. Erl, *Service-oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, 2005.

[3] M. Papazoglu, P. Traverso, S. Dustdar, and F. Leyman, "Service-oriented computing: State of the art and research challenges," *IEEE Computer*, Nov. 2007.

[4] *Web Services Description Language (WSDL) Version 2.0*, W3C Std. Recommendation 26 June 2007. [Online]. Available: http://www.w3.org/TR/wsdl20

[5] E. Giallonardo and E. Zimeo, "More semantics in QoS matching," in *Service-Oriented Computing and Applications, 2007. SOCA '07. IEEE International Conference on*, 2007, pp. 163–171. [Online]. Available: http://dx.doi.org/10.1109/SOCA.2007.30

[6] G. Dobson, R. Lock, and I. Sommerville, "QoSOnt: a QoS ontology for service-centric systems," in *Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE Computer Society, 2005, pp. 126–133.

[7] Y. Liu, A. H. Ngu, and L. Z. Zeng, "QoS computation and policing in dynamic web service selection," in *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. New York, NY, USA: ACM, 2004, pp. 66–73.

[8] G. Dobson, S. Hall, and G. Kotonya, "A domain-independent ontology for non-functional requirements," in *IEEE International Conference on E-Bussines Engineering*. IEEE Computer Society, 2007, pp. 563–566.

[9] Q. Ma, H. Wang, Y. Li, G. Xie, and F. Liu, "A semantic QoS-aware discovery framework for web services," in *ICWS*, 2008, pp. 129–136.

[10] I. Sora and D. Todinca, "Specification-based retrieval of software components through fuzzy inference," *Acta Politehnica Hungarica*, vol. 3, no. 3, pp. 121–132, 2006.

[11] C. Wu and E. Chang, "Intelligent web services selection based on AHP and Wiki," in *WI '07: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 767–770.

[12] H. Chan, T. Chieu, and T. Kwok, "Autonomic ranking and selection of web services by using single value decomposition technique," in *ICWS*, 2008, pp. 661–666.

[13] D. A. Menascé and V. Dubey, "Utility-based QoS brokering in service oriented architectures," in *ICWS*, 2007, pp. 422–430.

[14] L. Mei, W. Chan, and T. Tse, "An adaptive service selection approach to service composition," in *Web Services, IEEE International Conference on*. Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 70–77.

[15] X. Ye and R. Mounla, "A hybrid approach to QoS-aware service composition," in *ICWS*, 2008, pp. 62–69.

[16] V. X. Tran and H. Tsuji, "QoS based ranking for web services: Fuzzy approaches," in *Proceedings 4th International Conference on Next Generation Web Services Practices, NWESP '08*, Seul, Korea, Oct. 2008, pp. 77–82.

[17] P. Xiong and Y. Fan, "QoS-aware web service selection by a synthetic weight," in *Proceedings of the 4th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD (3)*, 2007, pp. 632–637.

[18] M.-F. Chen, T. Gwo-Hshiung, and C. Ding, "Fuzzy MCDM approach to select service provider," in *Proceedings IEEE International Conference on Fuzzy Systems*, 2003.

[19] M. De Cock, S. Chung, and O. Hafeez, "Selection of web services with imprecise QoS constraints," in *Web Intelligence, IEEE/WIC/ACM International Conference on*, 2007, pp. 535–541. [Online]. Available: http://dx.doi.org/10.1109/WI.2007.92

[20] P. Wang, K.-M. Chao, C.-C. Lo, C.-L. Huang, and Y. Li, "A fuzzy model for selection of QoS-aware web services," in *e-Business Engineering, 2006. ICEBE '06. IEEE International Conference on*, 2006, pp. 585–593.

[21] S. Agarwal and S. Lamparter, "User preference based automated selection of web service composition," in *Proceedings of the ICSOC Workshop on Dynamic Web Servicesses,*, 2005.

[22] ——, "Smart: A semantic matchmaking portal for electronic markets," in *E-Commerce Technology, IEEE International Conference on*. Los Alamitos, CA, USA: IEEE Computer Society, 2005, pp. 405–408.