Non-functional Parameters as First Class Citizens in Service Description and Matchmaking - An Integrated Approach

Mohamed Hamdy, Birgitta König-Ries, and Ulrich Küster

Institute of Computer Science, Friedrich-Schiller-Universität Jena,
07743 Jena, Germany
{hamdy,koenig,ukuester}@informatik.uni-jena.de

Abstract. Automatic discovery and invocation of services will only be accepted in practise, if the non-functional, especially QoS, parameters are taken into consideration during matchmaking. In this paper we present how this can be achieved. As an example, we use the DIANE[1] framework and explain how the concepts developed there can be extended to easily accommodate non-functional aspects. Thereby, these aspects become first class citizens given the same importance as functional requirements during service selection.

1 Introduction

Over the last few years, a number of proposals for semantic web service description languages and accompanying matchmakers have been developed. Virtually all of these languages foresee to describe a service along both its functional and non-functional parameters. However, likewise, virtually all of these approaches left the notion of non-functional parameter rather vague. Also, virtually all first-generation matchmakers concentrated on functional aspects of services only and did not take non-functional parameters into account at all.

If you look at realistic applications, e.g., today's applications of web services, it becomes clear, that this is not sufficient. There, non-functional aspects, in particular Quality of Service parameters, play a decisive role. Typically, a user will not consider to use a service – even if it offers exactly what she is looking for in terms of functionality – if it cannot meet her QoS requirements. Oftentimes, a requester will rather be willing to compromise on functional than on non-functional requirements.

Having recognized this, the semantic web services community has recently started to take a closer look at non-functional parameters and how to consider them during matchmaking. A discussion of this work can be found in the related works section. Basically, what these approaches do is the following: In a first step, services are selected based on the functionality requested. Subsequently, a filter is applied that eliminates all those service offers that do not meet the non-functional requirements.

[1] http://hnsp.inf-bb.uni-jena.de/DIANE/en/index.html

E. Di Nitto and M. Ripeanu (Eds.): ICSOC 2007 Workshops, LNCS 4907, pp. 93–104, 2009.

In this paper, we argue that such a separate consideration of functional and non-functional requirements is *not* the best way to approach the problem. First, the distinction between functional and non-functional attributes of a service is somewhat artificial and often arbitrary. Should *price*, for instance, be regarded as functional or non-functional? There are good arguments for either decision. Second, the two phase approach described above prevents the user to weigh functional against non-functional aspects. Consider, e.g., two offers: Both offers provide the possibility to download maps of German cities. Assume Offer A offers maps in a resolution of 1:10,000 over a rather slow network connection for a price of 0,50 Euro each. Offer B offers maps in a resolution of 1:15,000 over a fast connection for a price of 0,60 Euro each. The first question that arises is: What are functional, what are non-functional attributes here? The name of the city undoubtedly falls in the first category. But what about resolution and price? Both are sometimes tagged as functional, sometimes as non-functional. Let us for the moment assume they are non-functional. Based on the functional description alone, a user looking for high-resolution maps would clearly prefer offer A over offer B. However, if we take non-functional aspects into consideration, too, a hurried user may be willing to rather use service B, even if it is more expensive and offers lower quality maps. However, we can't be sure about this unless we give the user a possibility to specify her preferences in the request.

In the remainder of this paper, we will first take a look at what parameters are typically considered non-functional and will classify them into a number of different categories. For each of these categories we will then explain in detail if and how they can be handled by our semantic web services framework and what extensions are needed to fully support the integration of non-functional attributes.

2 Categories of Non Functional Attributes

In this section, we take a closer look at attributes that are often considered non-functional and will categorize them in three different classes:The three classes that we have identified are:

- *Static attributes*, e.g. price, resolution, print quality, ...
- *Dynamic attributes within the influence of the service provider*, e.g., price, printing time, time to get to the customer, ...
- *Dynamic attributes beyond the influence of the provider*, e.g., bandwidth, error rate, reputation, ...

2.1 Static Attributes

Static attributes are attributes whose value does not change over time. Such attributes can thus be static parts of the service description. Examples for static attributes are the countries a flight service offers flights to, the price per picture offered by a photo printing service, the resolution offered by a printer, the

delivery time guaranteed by a shipment company etc. Static attributes can be functional (e.g., the type of notebooks sold by an online trader) or non functional and are the easiest to model category of attributes.

2.2 Dynamic Attributes within the Influence of a Service Provider

For other attributes, the value changes over time. This may be due to choices a service user makes, but can also be due to the current state of the service provider or the environment. The value of the price attribute of an airline reservation service for instance will change depending on the destination the user chooses, but will also change over time for the same flight. In all these cases, the service provider will be able to determine the correct value of the attribute at any given point in time and will be able to provide the service user with this attribute value. One reason, why such dynamic attributes occur is that they offer a possibility to have configurable services, another reason is that the value for the same service really changes. These attributes can, again, be functional or non-functional and there should not be a need to distinct between the two.

2.3 Dynamic Attributes Beyond the Influence of a Service Provider

The third, and most challenging category of attributes are attributes that change over time and where the service provider does not necessarily know the value at any given point in time. Examples for such attributes are available bandwidth, reputation of the service provider, response time (including communication time) etc. This category contains most attributes typically characterized as QoS parameters. Even though one can construct some rather pathological examples (a hot air balloon floating around and sending a picture of the ground beneath it), typically, functional attributes do not fall into this category. We will later discuss, where to obtain values for these attributes from (as needed during matchmaking).

3 Modeling Non-functional Attributes in Service Offers

In this section, we describe how the different types of (non) functional attributes can be modeled in service offers. We base this description on our service description language, DIANE service descriptions (DSD).

3.1 An Introduction to DSD

Diane Service Descriptions (DSD) is a language specifically developed to semantically describe services. Thus one main difference between DSD and other semantic service description languages is DSD's own light-weight ontology language that is specialized for the characteristics of services and can be processed efficiently. The basis for this ontology language is standard object orientation which is extended by additional elements. In [1] we summarize these elements as follows:

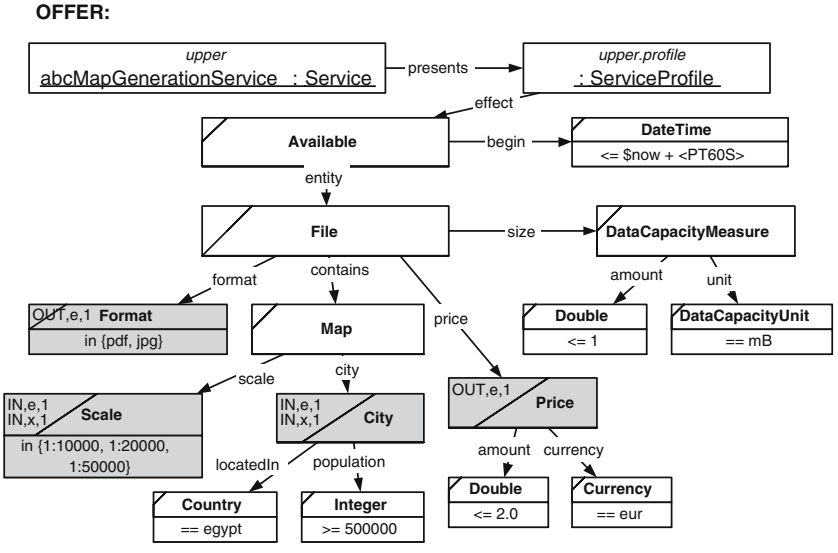

Fig. 1. An Example DSD offer

- Services perform world-altering operations (e.g., after invoking a shipment service, a package will be transported and a bill will be issued) which is captured by *operational elements*. We view this is the most central property of a service, thus, in DSD, services are described primarily by their effect on world, i.e. what they do – all other aspects (as flow of information, choreography etc.) are seen as secondary, derived properties. The effect of a service is comprehended as the achievement of a new *state* of the world, which in DSD is an instance from a state ontology.
- Services are typically able to offer not one specific effect, but a set of similar effects. An Internet shop for instance will be able to offer a variety of different products and will ship each order to at least all national addresses. That means, services offer to provide one out of a set of similar effects. Requesters on the other hand are typically accepting different services but with differing preference. A more expensive service might be as good as a less expensive one if it offers better QoS guarantees, whereas otherwise it might still be acceptable, but with lower preference. Thus, DSD is based on the notion of *sets*. Offers describe the set of possible effects they can provide and requests describe the set of effects they are willing to accept. Like all DSD concepts, sets are declaratively defined which leads to descriptions as trees as seen in Figure 1.
- *Valuing elements* are used to express preference for particular services over others within requests. In DSD, these elements are represented in requests by using fuzzy instead of crisp sets to describe a service. The larger the membership value the higher the preference of the requester. This allows fine-grained specification of preferences even in absence of a single optimization

goal (a service with high quality and high price might be just as preferred as a service with lower quality and lower price.)
- Service offers allow to choose among the offered effects (e.g. a shipment providers will allow to input the package being transported and to select where to pick it up and where to deliver it) which is captured by *selecting elements*. In DSD, selecting elements are represented as variables that can be integrated into set definitions, thus leading to configurable sets. Therefore, a service offer in DSD is represented by its effects as configurable sets of states.

In order to interact with a service, DSD assumes a simple choreography. During matchmaking several web safe *estimation operations* may be performed where operations of the service are called, which provide information but do not imply a contract between the provider and the client. After the best match is found that service can be invoked by executing a single *execution operation* which is supposed to produce the offered effects.

Figure 1 shows an example DSD offer for a service that allows to download maps of big cities in Egypt. The offer states that service execution will achieve an effect, namely that something becomes available. It will take at most one minute for the effect to occur (i.e., the response time is less than 60 seconds). The thing that becomes available is a file smaller than 1 MB in either pdf or jpg format that contains a map of an Egyptian city with more than 500,000 inhabitants. The scale of the map is either 1:10,000, 1:20,000, or 1:50,000. The user will have to pay at most 2,50 Euros for the file. The price and the format are provided as output variables of the service; city and scale are input variables. The output variables are filled during the estimation phase, input variables need to be provided both at estimation and execution time.

3.2 Capturing Static Attributes

Expressing static non functional attributes works like expressing functional properties of a service and is very straightforward. The example in Figure 1 shows an offer, that has, for instance, a static *size* attribute. For this and other static attributes, values can be determined at the time of setting up a service description.

3.3 Capturing Dynamic Attributes within the Influence of the Service Provider

Often times providers will need to include dynamic attributes into their service descriptions. On the one hand, service offers will typically describe not a single service, but a set of services. Our example service, e.g., does not offer exactly one map, but one out of a set of maps. When a user is interested in the offer, she will choose one of these services. Depending on which of the offered services is chosen, attribute values will differ. The concrete value for the price attribute, e.g., can only be determined once the concrete map has been chosen. All the

provider can do at the time of description is to restrict the range of possible values (in the example, the provider guarantees, that no map more expensive than 2,50 Euros will be made available).

On the other hand, attribute values may depend on the state of the provider or the environment. For instance, the response time of a service may depend on its current load or the price of a ticket may depend on current exchange rates. In order to give a valid upper bound on the response time or the price, providers would have to state the worst expected time/price in the static description.

In both cases it is thus desirable to equip providers with means to offer such information in a more dynamic fashion. This is supported by DIANE through the aforementioned estimation operations. By tagging a concept as estimation out variable (`OUT,e,i`), providers can declare that the value of that concept may be dynamically inquired by executing the *i*th estimation operation, providing the values of the concepts tagged as `IN,e,i` as input. In the example the concrete value of the *city* and *scale* attributes need to be filled in at estimation time, in turn, the service provider returns *price* and *format* information.

The DSD matchmaker will use a multi-phased approach to integrate estimation operations into the matchmaking process. In a first phase the matchmaker will determine concrete input values to use for the execution of an estimation operation. It will also collect information about whether an estimation operation of an offer at hand is *promising*, i.e. whether the value provided by that operation will be helpful at all to decide how well that offer matches a given request. In a second phase the matchmaker will then execute only the promising estimation operations and update the corresponding service offer descriptions with the dynamically inquired information. In a third phase the final matchmaking is performed based on the updated descriptions. Further information on how to integrate dynamic information into service descriptions can be found in [2].

3.4 Capturing Dynamic Attributes Beyond the Influence of the Service Provider

Above and so far in our example, we have assumed, that the response time may vary but is under the control of the service provider. More realistically, response time depends on the service provider as well as external conditions beyond his control - and possibly also beyond his knowledge. This is the most difficult to model case. DSD offers basically two ways to address such attributes: First, we can model them just like dynamic attributes under the influence of the service provider. In this case, we assume that the provider has a means to obtain the necessary information when executing the estimation steps and that he will then fill in the attribute values accordingly. In the case of response time, such a means could be a monitoring program run by the service provider or an external monitoring service that the service provider calls in turn when he needs the respective information. In both cases, from the point of view of the matchmaker, these attributes are treated just like regular dynamic attributes.

Another possibility would be to exclude them completely from the offer descriptions and to use service composition to obtain the necessary information. We will take a closer look at this later on when discussing matchmaking.

4 Modeling Non Functional Requirements in Service Requests

From the point of view of a request, there is no difference between a functional and a non-functional requirement. As depicted in Figure 2 request descriptions are pretty similar to offer descriptions. There is one decisive difference, however: Service requests can contain fuzzy sets. These are used to encode preferences among a number of options.

Our example requester wants to obtain a map of Cairo within the next 90 seconds. He is willing to accept maps in the two listed scales but prefers the first scale towards the second one (the given value [0.7] encodes a 70% match as opposed to [0.3] which encodes a 30% match). He is willing to pay at most 2.50 Euros for the map, but prefers cheaper ones. $\sim<$ [2.50] 0 means that (unrealistic) prizes below zero Euros will have a membership value of 1 in the fuzzy set, values above 2.50 will have a membership value of 0 and values in between will follow a linearly decreasing function. Also, the service provider should be trustworthy. The key difficulty here is how to balance different optimization goals. It is clear that the perfect service will match perfectly in terms of functional/non functional properties, will be very inexpensive and will have optimal QoS guarantees. In reality, of course, there will be services with low QoS guarantees but also low cost, services with good QoS but high cost, services that match functionally very well but do not provide QoS guarantees at all and so on. In such a setting services with quite different characteristics might be equally preferable to the requester (e.g. an inexpensive service with low QoS guarantees versus an expensive one with high QoS guarantees). Furthermore QoS and functional matching cannot be treated separately either. A service that is less preferable in terms of its functionality might be the best choice overall if it matches very well in terms of non-functional characteristics and vice versa. Maybe the requester is particularly interested in some functional and some non functional characteristics and is willing to compromise with respect to others.

Since DSD does not distinguish between non functional and functional requirements, this can be easily achieved. On the one hand, the fuzzy sets explained above allow to describe preferences for individual attribute values. Connecting strategies are then used to express how to weigh the preferences against each other. Our example uses the default connecting strategy, i.e. the membership value of an instance of an inner set is determined by multiplying the membership values of the child sets. Assume, for instance, a file containing a map of Cairo in 1:20,000 for the price of 1.25 Euro. The respective instance has a membership value of 0.3 in the *map* set (1 for Cairo times 0.3 for the scale). For price, the membership value is 0.5, so that overall the membership value is 0.3 * 0.5 = 0,15. This default strategy should be used, if all attributes are deemed equally

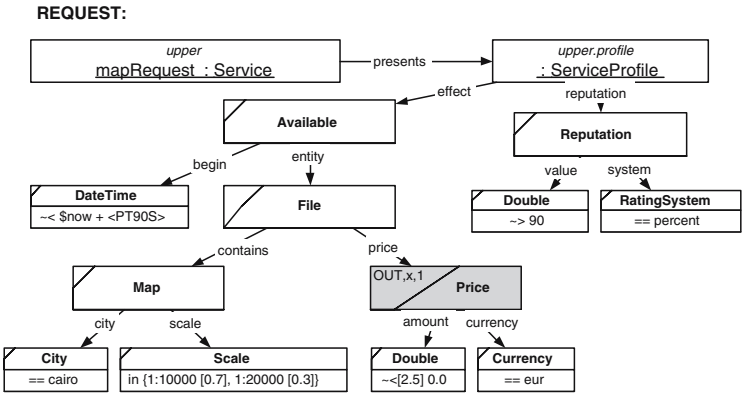

Fig. 2. A sample request

important. If, on the other hand, the requester is more interested in obtaining a map in the right scale and not so interested in the price, a different connecting strategy, e.g., a weighted sum could be used. For a complete description of connecting strategies, please refer to [3].

Overall, this offers a very powerful means for the user to precisely model preferences without the need to prioritize functional over non functional requirements or the other way round.

5 Using Non Functional Specifications in Matchmaking

In this section, we give a brief overview of the current DSD matchmaker. We will then discuss if and if so, how it needs to be extended to deal with the three categories of non functional properties. The DSD matcher has been thoroughly described in [1,4]. Let us briefly summarize how this component works: For a given DSD offer description o and a given DSD request r, a matchmaker has to solve the following problem: *What configuration of o's effect sets is necessary to get the best fitting subset of r's fuzzy effect sets.*

To obtain this configuration, our matchmaker traverses the descriptions of the request and the offer at hand in parallel and recursively compares corresponding concepts (remember that DSD descriptions largely form trees). The degree of match (matchvalue) of two particular concepts is determined by applying any direct restrictions to that concept from the request and combining the matchvalue of the type of the concepts with the matchvalues of the properties of the concepts. These are determined in a recursive fashion. When the matchmaker encounters a variable in the offer during its traversal of the descriptions it determines the optimal value for this variable with respect to the request. Also, the request `out` variables are bound. The matchmaker has been extensively evaluated both in our own experiments and within the Semantic Web Services Challenge[2] and has

[2] http://sws-challenge.org

proven to be both powerful and efficient. Note that in our work we assume that requests and offers are described using the same ontologies and do not deal with mediators. However, the concept of mediators is orthogonal to the work in this paper and the use of mediators could be integrated in a straightforward way into the DSD matchmaking algorithm.

Currently the matcher does not offer a specific treatment of non-functional attributes. Let us take a closer look at our three categories of attributes and see whether the matchmaker can successfully deal with them:

It should be obvious that static attributes do not pose any problems. What about dynamic attributes under the influence of the service provider? Examples for such attributes are price, city and scale. Both possibly non-functional attributes like price and scale and definitely functional attributes like city can be treated effortlessly by the matchmaker. Since, e.g., Cairo is an Egyptian City with more than 500,000 inhabitants, the matchvalue for the *city* set will be 1. The concrete match value for the price will depend on the value returned by the service provider during the estimation phase. Based on the static information given, we can be sure that it will be greater than 0 (unless no suitable map is offered at all).

For the response time (encoded in the *begin* attribute of the *available* set), the matchmaker can statically determine a match, since all possible provider values are members of the request set. Assume for a moment, that the requester has specified response time as an `out` variable. In this case, it needs to be treated during estimation. For the matchmaker it is transparent how the service provider determines the appropriate value. In particular, the service provider could in turn call a network monitoring service to determine current network load before estimating the response time. Thus, dynamic attributes both within and beyond the influence of the service provider can be treated as long as the service provider is willing to procure the appropriate values.

If you look at offer and request once more, you will notice that the reputation attribute does not appear at all in the service offer. Currently, the matchmaker would compute that part of the desired effect can not be met by the offer (or at least it cannot be determined from the offer description whether it provides the effect or not). The matchmaker would therefore consider this offer and request not to be matching. On the other hand, as discussed earlier, it does not make sense to include attributes like reputation into the offer description. If we want to be able to support such attributes – or more generally attributes of the third category where the service provider does not procure the values – extensions to the matchmaking algorithm are needed. We discuss possible solutions in the next section.

Keep in mind, however, that we can seamlessly integrate the vast majority of non functional requirements into our matchmaking process without the need to distinguish between functional and non functional attributes and without the need of any extensions to either our language or the matchmaking component.

6 Challenges with Respect to Non-functional Properties

As described in the last section, the only case where we need to extend our matchmaker is to support attributes of the third category that are not procured by the service provider. We envision two possible approaches to this problem. Both are based on existing extensions of DSD respectively the matchmaker that were not introduced in the last section. None of these extensions is yet able to handle the case described above, but they should offer a good basis for a solution.

Let us first briefly introduce these extensions and then explain how they could be adapted to handle the problem at hand. Consider as an example a more complex request, where a user wants to book a flight to a city in Egypt and also wants to obtain a map of the destination city of his flight. Additionally, he specifies a maximum total price. In order to express conditions that span several attributes, like total price, we have added the possibility to name variables and to add conditions on these variables to the service profile. The named variables are also used to connect the two effects, i.e., to make sure that the map is a map of the flight destination.

The matcher has been extended to allow for service composition when the achievement of several effects is involved. The extended matcher will find individual service offers that provide both effects (a flight and a map), but also combinations of two service providers, where one offers a flight and the other one a map. The matchmaker will configure these services to ensure that the conditions on the variables are met.

How can these extensions be used to handle our problem? The first approach would be to extend the possibility to add conditions to the service profile beyond conditions on attributes. We could add there a condition on the reputation of the service provider. The second possibility would be to offer a mechanism to rewrite requests containing this type of attribute into two new requests: Request 1 should contain the original request without the problematic attribute, Request 2 should be a request where the effect should be that something, e.g., the reputation of the service provider, becomes known. Both requests then need to be coupled via common variable names.

The second request could either be a regular request where the matchmaker determines an appropriate service provider or could be tagged as a request to be handled by an internal service of the requester, e.g., the requester's instance of a distributed reputation system. We are currently investigating which of these two options is more desirable. Both extensions are pretty straightforward on a conceptual level but require some implementation effort.

7 Related Work

While all semantic service description languages allow to specify non functional parameters, it has been only recently that these parameters start to be used in matchmaking. Probably the first work to address the need to take non functional requirements into consideration is [5]. The paper argues why QoS needs

to be considered and introduces a model to integrate these parameters into the service selection process. [6] describes a QoS ontology based on DAML-S and introduces a matchmaking algorithm for QoS specifications. The matchmaker works analogous to the one described in [7] for functional descriptions. [8,9,10] consider extensions for WSMO. They introduce WSML QoS ontologies and describe selection processes based on these QoS factors. [11] also introduces a QoS model that is then used for interactive choice of services, i.e., users are supported in finding the best matching service but no automatic matchmaking is performed. For our own work, the ontologies developed in these and other approaches are very valuable and can be seamlessly integrated (after a translation to our ontology language).

In contrast to our work, however, virtually all of these proposals take a two step approach to matchmaking: In a first step, services that meet the functional requirements of the service requester are chosen. In a second step, non functional requirements are then used as a filter to select the most appropriate of these services. The assumption here is that the first step will return a set of functionally equivalent services that perfectly match the user's request. In our opinion, this approach has two major drawbacks: First, it forces a distinction between functional and non functional attributes that is not always trivial to make. Second, the assumption that there will be a large set of functionally completely equivalent services that happen to offer exactly what the user wants is unrealistic for all but very simplistic cases. Much more frequently, the user will be somewhat flexible in his requirements and the service providers will offer services that differ slightly. Here, an approach that is a lot more flexible than the ones proposes up to now, is needed.

8 Summary and Conclusion

In this paper, we have argued, that the best way to incorporate non functional parameters into service descriptions and matchmaking is to simply ignore the distinction between functional and non functional parameters and to provide a uniform treatment of both. We have shown how this can be achieved in our DIANE framework. While it is very straightforward for the description part, where virtually no difficulties amount, it is a bit more complicated for the matchmaking part. There, too, most cases can be handled effortlessly with no or little extension of the existing matchmaker. The only case that we cannot yet handle are those non functional attributes that cannot be provided by the service offerer and are thus not part of his description. This has two reasons: On the one hand, the service offerer may not possess the necessary knowledge, e.g., about the current network load to correctly estimate, e.g., response time. On the other hand, the service provider may simply not be the right choice to ask for certain information, e.g., with respect to his own trustworthiness. In this paper, we propose two possible extensions to our framework to handle this case: Adding conditions on the service provisioning to the service profile in the request or rewriting requests and then using service composition. We hope to be able to provide an evaluation

of the approach in the near future, preferably within the SWS Challenge, which we have successfully used to evaluate previous work already.

References

1. Küster, U., König-Ries, B., Klein, M., Stern, M.: DIANE - a matchmaking-centered framework for automated service discovery, composition, binding and invocation. International Journal of Electronic Commerce (IJEC) Special Issue on Semantic Matchmaking and Retrieval (2007)
2. Küster, U., König-Ries, B.: Supporting dynamics in service descriptions - the key to automatic service usage. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 220–232. Springer, Heidelberg (2007)
3. Klein, M., König-Ries, B., Müssig, M.: What is needed for semantic service descriptions - a proposal for suitable language constructs. International Journal on Web and Grid Services (IJWGS) 1(3/4), 328–364 (2005)
4. Klein, M., König-Ries, B.: Coupled signature and specification matching for automatic service binding. In: Zhang, L.-J., Jeckle, M. (eds.) ECOWS 2004. LNCS, vol. 3250, pp. 183–197. Springer, Heidelberg (2004)
5. Ran, S.: A model for web services discovery with QoS. ACM SIGecom Exchanges 4(1), 1–10 (2003)
6. Zhou, C., Chia, L.-T., Lee, B.-S.: DAML-QoS ontology for web services. In: Proceedings of the 2004 IEEE International Conference on Web Services (ICWS 2004), San Diego, California, USA (July 2004)
7. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.P.: Semantic matching of web services capabilities. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342. Springer, Heidelberg (2002)
8. Wang, X., Vitvar, T., Kerrigan, M., Toma, L.: A qoS-aware selection model for semantic web services. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 390–401. Springer, Heidelberg (2006)
9. Liu, Y., Ngu, A., Zheng, L.: QoS computation and policing in dynamic web service selection. In: Proceedings of the 13th International World Wide Web Conference (WWW 2004), Manhattan, NY, USA (2004)
10. Toma, I., Roman, D., Fensel, D.: A multi-criteria service ranking approach based on non-functional properties rules evaluation. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 435–441. Springer, Heidelberg (2007)
11. Yu-jie, M., Jian, C., Shen-shen, Z., Jian-hong, Z.: Interactive web service choice-making based on extended QoS model. In: Proceedings of the Fifth International Conference on Computer and Information Technology (CIT 2005) (September 2005)