A Service Ranker based on Logic Rules Evaluation and Constraint Programming

José María García[1], Ioan Toma[2], David Ruiz[1], and Antonio Ruiz-Cortés[1⋆]

[1] University of Sevilla
E.T.S. Ing. Informática, Av. Reina Mercedes s/n, 41012 Sevilla, Spain
{josemgarcia, druiz, aruiz}@us.es
[2] Semantics Technology Institute - STI Innsbruck, University of Innsbruck
Technikerstrasse 21a, A-6020 Innsbruck, Austria
ioan.toma@sti2.at

Abstract. Ranking of Semantic Web Services is usually performed based on user preferences descriptions. These descriptions are expressed in terms of an underlying logical formalism, which limits their expressiveness. Thus, there are some kind of descriptions, such as utility functions, that cannot be handled by reasoners currently being used to perform Semantic Web Services tasks, though utility functions provide a higher level of expressiveness. In this work, we present a hybrid solution to allow the introduction of utility functions in user preferences descriptions, using both Logic Programming rules evaluation and Constraint Programming to perform the ranking process. This proposal is based on the Web Service Modeling Ontology, extending it with a highly expressive framework to specify user preferences, and enabling the integration of different engines to perform the ranking process.

Keywords: Semantic Web Services, Ranking, Non-Functional Properties, Rules Evaluation, Constraint Programming.

1 Introduction

Semantic Web Services (SWS) technologies enable the automatization of service related tasks, such as discovery, ranking, and selection. In particular, discovery of services that fulfill certain user requirements have been widely treated in several proposals, such as [3, 5, 6, 12]. These proposals are mostly based on Description Logics reasoners, which match service descriptions with user requirements. These discovered services need to be ranked in order to select the best service according to stated user preferences.

Ranking and selection proposals use specific descriptions to perform these tasks. These descriptions, i.e. user preferences, are based on non-functional properties of services, which specify preferences with respect to properties that cannot be considered

⋆ This work is partially supported by the European Commission (FEDER) and Spanish Government under CICYT project Web-Factories (TIN2006-00472), by the Andalusian Government under project ISABEL (TIC-2533), and by the EU FP7 IST project 27867, SOA4ALL - Service Oriented Architectures For All.

functional or behavioral, such as price, security, reliability, etc. Thus, discovered services are ranked and selected depending on their non-functional properties. Although there are several proposals that provide a semantic framework to express these properties in a selection scenario, such as [7, 15, 17], preferences can only be expressed using tendencies or order conditions. Other proposals use utility functions to describe preferences [4, 9], which provide higher expressiveness and make use of Constraint Programming (CP). However, those approaches do not provide a semantic framework to define these utility functions.

Our proposal presents a hybrid architecture to perform service ranking integrated in the Web Service Modeling Ontology (WSMO)[8]. Thus, user preferences are modeled using the Web Service Modeling Language (WSML)[11], adding support to utility functions. The proposed architecture integrate a reasoning engine that support rules evaluation and a CP solver. Our hybrid approach decouples user preferences descriptions with the engines that perform the ranking, so these engines can be interchanged. This also allows a high expressiveness when describing preferences, by means of utility functions.

The rest of the paper is structured as follows: Section 2 presents how services descriptions and user preferences are modeled using utility functions within WSMO and its language WSML. Then, in Sec. 3 the proposed architecture for service ranking is introduced, describing its components and implementation requirements. A run-through scenario is described in Sec. 4 for further illustration of the proposed architecture. Related work is discussed in Sec. 5. Finally, Sec. 6 sums up our contribution and outlines further research that should be addressed.

2 Modeling Approach

In this section we briefly introduce our approach for modeling non-functional properties of services and user preferences. We use WSMO and WSML to model services and user requests. We are mainly interested in modeling non-functional properties perspectives of service providers and service requestors. The rest of the section provides modeling details for both service descriptions (Sec. 2.1) and user requests and preferences (Sec. 2.2).

2.1 Service Descriptions

In WSML[11], non-functional properties are modeled in a way similar to which capabilities could be currently modeled in WSML. Non-functional properties are defined using logical expressions same as pre/post-conditions, assumptions and effects are being defined in a capability. The terminology needed to construct the logical expressions is provided by non-functional properties ontologies (c.f. [13]).

For exemplification purposes we use the SWS Challenge[3] Shipment Discovery scenario. We are mainly interested in two aspects of shipment services for this particular

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

scenario, namely discounts (pricing) and obligations. The shipping services allows requestors to order a shipment by specifying, sender's address, receiver's address, package information and a collection interval during which the shipper will come to collect the package.

Listing 1.1 displays a concrete example on how to describe one non-functional property of a service (i.e. Runner), namely obligations. Due to space limitations the listing contains only the specification of obligations aspects without any functional, behavioral or any other non-functional descriptions of the service. In an informal manner, the service obligations can be summarized as follows: (1) in case the package is lost or damaged Runner's liability is the declared value of the package but no more than 150$ and (2) packages containing glassware, antiques or jewelry are limited to a maximum declared value of 100$.

Listing 1.1. Runner's obligations

```
namespace {_"WSRunner.wsml#",
    runner _"WSRunner.wsml#", so _"Shipment.wsml#",
    wsml _"http://www.wsmo.org/wsml/wsml−syntax/",
    up _"UpperOnto.wsml#"}

webService runnerService
    nfp
        up#hasObligations hasValue runner#DefinitionObligations
        up#hasObligationsFunction hasValue runner#hasPackageLiability
    endnfp
    nonFunctionalProperty DefinitionObligations
     definition
      definedBy
        //in case the package is lost or damaged Runner's liability is
        //the declared value of the package but no more than 150 USD
        hasPackageLiability(?package, 150):− ?package[so#packageStatus hasValue ?status] and
          (?status = so#packageDamaged or ?status = so#packageLost) and
          packageDeclaredValue(?package, ?value) and ?value>150.

        hasPackageLiability(?package, ?value):− ?package[so#packageStatus hasValue ?status] and
            (?status = so#packageDamaged or ?status = so#packageLost) and
            packageDeclaredValue(?package, ?value) and ?value =< 150.

        //in case the package is not lost or damaged Runner's liability is 0
        hasPackageLiability(?package, 0):− ?package[so#packageStatus hasValue ?status] and
        ?status != so#packageDamaged and ?status != so#packageLost.

        //packages containing glassware, antiques or jewelry
        //are limited to a maximum declared value of 100 USD
        packageDeclaredValue(?package, 100):−
          ?package[so#containesItemsOfType hasValue ?type, so#declaredValue hasValue ?value] and
          (?type = so#Antiques or ?type = so#Glassware or ?type = so#Jewelry) and ?value>100.

        packageDeclaredValue(?package, ?value):−
          ?package[so#containesItemsOfType hasValue ?type, so#declaredValue hasValue ?value] and
          ((?type != so#Antiques and ?type != so#Glassware and ?type != so#Jewelry) or ?value<100).

    capability runnerOrderSystemCapability
    interface runnerOrderSystemInterface
```

Runner's obligations are expressed as logical rules in WSML. Additionally, it is necessary to explicitly define which predicate is actually being used to obtain the value of the Runner's obligations, by means of the `hasObligationsFunction` property. Similarly other non-functional properties can be encoded using WSML rules.

2.2 User Preferences

User preferences express how important certain non-functional properties are from the service user's point of view. Thus, preferences are taken into account when performing ranking tasks. Utility functions are a highly expressive formalism to describe user preferences. An utility function is defined as a normalized function (ranging over $[0, 1]$) whose domain is a non-functional property, giving information about the preferred range of values for that non-functional property. Figure 1 shows two utility functions defined as piecewise functions. On the one hand, lower price values are preferred. Thus, the highest utility value is returned by that function if price is below 60 dollars, decreasing that value linearly until 300 dollars, where the utility is at its minimum. On the other hand, the user prefers higher obligations values, so the utility function is modeled as shown in Fig. 1, varying from the minimum utility value (0) when the liability value is below 50, and growing linearly until liability reaches 130, where the utility is maximum (1).

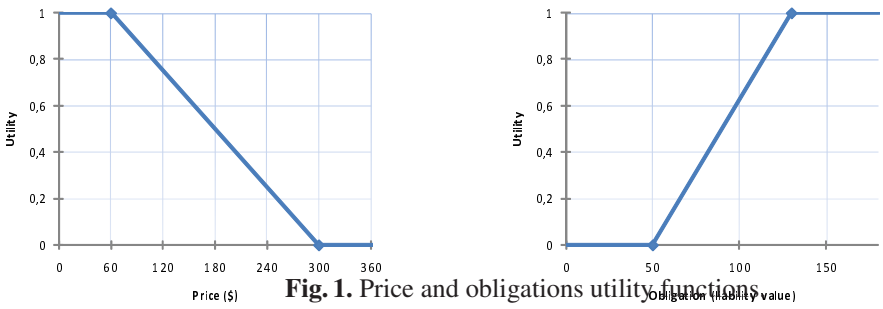

Fig. 1. Price and obligations utility functions

In order to express user preferences combining several non-functional properties, each utility function has to be associated with a relative weight. Thus, in a multi-criteria ranking process, the user preference value that is used to rank services is a weighted composition of the associated utility function values. That user preferences definitions are included as a part of a goal. For instance, from the goal description shown before, a user may want to rank services with respect to the utility functions for price and obligations from Fig. 1, with associated weights of 0.6 and 0.4, respectively.

Listing 1.2 contains the WSML encoding of the previous presented user preferences. Such preferences are expressed as part of the user request that we call goal. Preferences are encoded as WSML rules which enables reasoning which in turn provides support for service related tasks in which preferences are considered (e.g. discovery, selection, and ranking). As discussed later in Sec. 3, besides the actual reasoning with WSML rules, a translation of the user preferences expressed as rules to CP is provided. This enables the use of utility functions for modeling preferences which provide high expressiveness.

To model preferences in WSML we defined a binary predicate `hasPreference` that takes as first argument the identifier of a non-functional property and as second argument the preferred value of that non-functional property. For each interval in the domain of preference function, a WSML rule is defined. At runtime, namely during ranking process, one of the rules will fire and the value of the preference function is determined. Additionally, another binary predicate `hasComplexPreference` is used in cases where the preferred value of the non-functional property identified in its first argument is computed using WSML built-in numeric functions.

Listing 1.2. Goal description with preferences encoded in WSML

```
namespace {_"Goal1.wsml#", req _"Goal1.wsml#", so _"Shipment.wsml#",
  dc _"http://purl.org/dc/elements/1.1#",
  wsml _"http://www.wsmo.org/wsml/wsml-syntax/",
  pref _"http://www.wsmo.org/ontologies/nfp/Preferences.wsml#",
  up _"http://www.wsmo.org/ontologies/nfp/upperOnto.wsml#"}

goal Goal1
 nfp
    up#order hasValue pref#ascending
    up#nfp hasValue {up#hasObligations, up#hasPrice}
    up#nfpFunction hasValue {up#hasObligationsFunction, up#hasPriceFunction}
    up#instances hasValue req#GumblePackage
    up#hasPreference hasValue req#DefinitionPreferences
    up#hasWeights hasValue req#DefinitionWeights
 endnfp

 capability requestedCapability
  postcondition
   definedBy
    ?order[so#to hasValue Gumble,so#packages hasValue GumblePackage] memberOf so#ShipmentOrder and
    Gumble[so#firstName hasValue "Barney", so#lastName hasValue "Gumble",
    so#address hasValue GumbleAddress] memberOf so#ContactInfo and
    GumbleAddress[ so#streetAddress hasValue "320 East 79th Street",
    so#city hasValue so#NY, so#country hasValue so#US] memberOf so#Address.

ontology requestOntology

instance GumblePackage memberOf so#Package
  so#length hasValue 10
  so#width hasValue 2
  so#height hasValue 3
  so#weight hasValue 10
  so#declaredValue hasValue 150
  so#containesItemsOfType hasValue so#Glassware
  so#packageStatus hasValue so#packageLost

axiom DefinitionPreferences
  definedBy
        hasPreference(up#hasObligations, 100):-
         up#hasObligations[value hasValue ?hasObligationsValue] and ?hasObligationsValue >= 130.

        hasPreference(up#hasObligations, 0):-
         up#hasObligations[value hasValue ?hasObligationsValue] and ?hasObligationsValue < 50.

        hasComplexPreference(up#hasObligations, ?obligationsPreferenceValue) :-
         up#hasObligations[value hasValue ?hasObligationsValue] and ?hasObligationsValue < 130 and
         ?hasObligationsValue >= 50 and ?obligationsPreferenceValue=((10*?hasObligationsValue-500)/8).

        hasPreference(up#hasPrice, 100):-
         up#hasPrice[value hasValue ?hasPriceValue] and ?hasPriceValue < 60.

        hasPreference(up#hasPrice, 0):-
         up#hasPrice[value hasValue ?hasPriceValue] and ?hasPriceValue >= 300.
```

```
        hasComplexPreference(up#hasPrice, ?pricePreferenceValue) :−
          up#hasPrice[value hasValue ?hasPriceValue] and ?hasPriceValue < 300
          and ?hasPriceValue >= 60 and ?pricePreferenceValue=((3000−10∗?hasPriceValue)/24).

axiom DefinitionWeights
  definedBy
        hasWeight(up#hasObligations, 40).
        hasWeight(up#hasPrice, 60).
```

3 A Hybrid Architecture for Service Ranking

Having modeled the service non-functional properties and user requests and preferences as described in Sec.2, we provide in this section a service ranking approach that uses hybrid descriptions of services and user requests, i.e. a combination of Logic Programming (LP) Rules and Constraint Programming (CP). LP Rules expressed as WSML logical expressions/axioms are mainly used to model services descriptions. User requests and preferences in terms of non-functional properties are expressed using a combination of LP Rules and CP formulas, being encoded with the use of the same WSML logical expressions/axioms. These expressions allow to define different kinds of utility functions.

To handle WSML logical expressions/axioms we use the IRIS[4] reasoner. In fact, any other reasoner that can handle WSML rules evaluation could be used, e.g. KAON2[5] or MINS[6], provided that it is integrated using the WSML2Reasoner framework [10]. For the CP part the Choco[7] system is used. The overall envision architecture is provided in Fig. 2.

As depicted in Fig. 2, the architecture of the hybrid ranking system contains a set of loosely coupled components. The user submits a request formalized as a WSML goal through an *Access interface* component. The request is formalized as presented in Sec. 2. Once submitted to the system the request is processed by the *Extractor* component. The job of the Extractor component is to parse the given request, and to identify the requested non-functional properties and their weights. Each non-functional property has an associated weight (i.e. numerical value) which gives the importance of non-functional property in user's view, relative to the other requested non-functional properties. The weights are encoded using a predefined axiom `DefinitionWeights`. This axiom is being parsed and a matrix of non-functional properties and weights is created by the *Extractor* component.

The *Evaluator* component is responsible for the evaluation of WSML rules that are used to encode the non-functional properties of the services. Only the rules/axioms that encode the non-functional properties requested by the user are evaluated. In the current prototype we use the IRIS reasoner. IRIS is an extensible reasoning engine for expressive rule-based languages that supports safe and un-safe datalog, negation as failure, function symbols, support for XML data types, and built-in predicates. The

[4] http://sourceforge.net/projects/iris-reasoner/
[5] http://kaon2.semanticweb.org/
[6] http://dev1.deri.at/mins/
[7] http://choco-solver.net/index.php?title=Main_Page

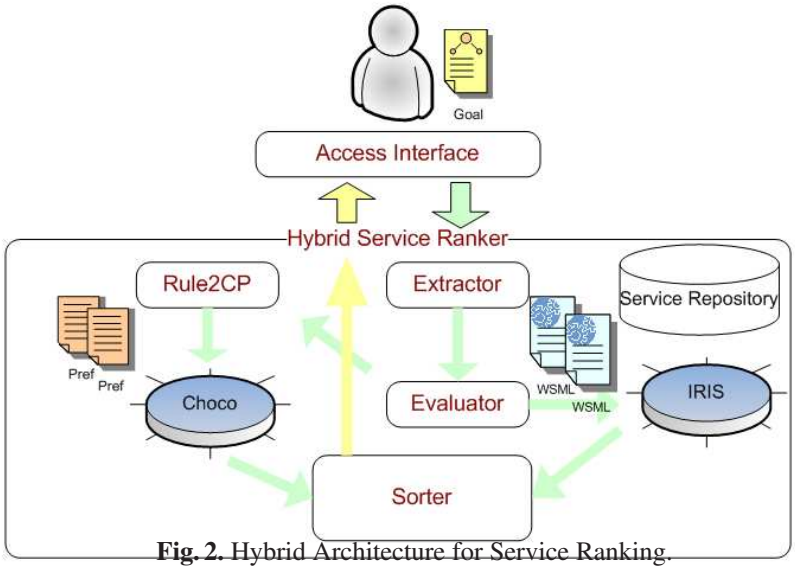

Fig. 2. Hybrid Architecture for Service Ranking.

overall process is based on our previous work described in [14]. In a nutshell each of the rules corresponding to a non-functional property of a service is evaluated, and the values obtained are normalized.

For the goal evaluation we use a CP approach to evaluate user preferences. More precisely, user preferences that are formalized as WSML rules are being translated to CP format using the *Rule2CP* component. The translated representation is evaluated using the Choco implementation. Choco is a Java library that allows the modeling of classical constraint satisfaction problems, optimization, scheduling and explanation-based CP, programmatically. During this evaluation step the rank values corresponding to each service that were evaluated using the IRIS reasoner are being used in the CP evaluation. Additionally, for each service, we perform an aggregation of the weighted non-functional properties values.

Finally the associated ranking values for each service are ordered and the ranked list of services is retuned to the user. The ordered list is being constructed by the *Sorter* computer. This list can be used as the input for the services selection process.

The integration of both main components, i.e. IRIS and Choco, allows to separate different stages of the ranking process where each stage is performed by the optimal engine. Thus, the IRIS reasoner initially evaluates rules about non-functional properties so their actual values are obtained for each candidate service to be ranked. The role of the Choco system is to solve a Constraint Satisfaction Optimization Problem (CSOP) that is obtained from the WSML axioms of the goal. Using this formalism, it is possible to define and better handle complex expressions and utility functions, which are then used for the service ranking process. Therefore, the optimization stage of the ranking

process is performed by a more suited system to solve the corresponding CSOP, i.e. Choco as opposite of IRIS.

4 A Use Case Scenario

Using the Shipment Discovery scenario shown in Sec. 2, a run-through of the prototype implemented is presented in the following. This prototype has been developed following the architecture introduced in Sec. 3, so its main purpose is precisely to evaluate the soundness of our architecture.

First of all, our ranker loads the user goal that contains the user preferences (cf. Listing 1.2). Previously, in a whole discovery scenario, a set of service descriptions that match the requested capability from the goal, must have been discovered. Thus, our ranker also loads the discovered service descriptions, as the excerpt shown in Listing 1.1. In both goal and service descriptions, information about which non-functional properties are going to be used to rank services is extracted by the extractor component. In this case, using the `nfp` and `nfpFunction` properties from the goal in our use case scenario, the extractor considers `hasObligations` and `hasPrice` as the non-functional properties to use for the ranking process. Correspondingly, the concrete predicates within each service description that are used to evaluate the associated non-functional property value are extracted from `hasObligationsFunction` and `hasPriceFunction` properties of those service descriptions.

Once the non-functional properties and their corresponding evaluation predicates are extracted from the loaded descriptions, each property is evaluated using IRIS. Thus, a query for each pair property-service is built, storing the resulting value for the next stage of the process. These values are passed to the *Rule2CP* component in order to incorporate them into the generated CSOPs.

This translator component takes preference definitions from the goal and non-functional properties values previously computed and parses them, generating the corresponding CSOPs. In order to perform that parsing, `hasWeight`, `hasPreference`, and `hasComplexPreference` predicates from the goal are traversed so the CSOP can be built incrementally using the Choco library. For each service, a different CSOP is created using its corresponding values previously stored. Thus, the Choco component is invoked for each generated CSOP, returning the global preference value for each service being ranked.

Finally, global preference values are sorted by the *Sorter* component, using an ascending order, as it is stated in the `order` property of the goal. Thus, previously discovered services are finally ranked in terms of their global preference values computed using our hybrid service ranker. This sorted list of services can be used by a selection component, so the best service that fulfills the user goal (including capabilities and preferences) is finally selected.

5 Related Work

There are some ranking and selection proposals that are based on non-functional properties. Thus, Pathak *et al.* use domain specific ontologies so services are ranked de-

pending on matching degrees and weighted functions [7]. Similarly, Zhou *et al.* rank services using matching degrees and provide an extension to DAML-S in order to include quality-of-service profiles [17]. Concerning WSMO, there is also an extension proposed by Wang *et al.* that define a ranking model based on a quality matrix, where user preferences are defined in terms of preferred tendencies and weights between each non-functional property [15]. Another WSMO extension, which our proposal is based on, is proposed in [14], where a multi-criteria ranking approach is presented.

Although not specifically focused on ranking services, [16] presents an approach where service composition is optimized using defined utility functions within an Integer Programming algorithm. Utility functions are also used to express user preferences in [9]. In that work, Ruiz-Cortés *et al.* perform the ranking using CP. This paradigm is also used in [4], where an ontology of non-functional properties is introduced to define them, though user preferences are not semantically defined. In [1] a generic hybrid model to perform discovery, ranking and selection is proposed, which is contextualized to WSMO in this paper. Moreover, in [2] a preliminary version of this work is presented.

6 Conclusions and Future Work

In this work, a SWS ranking proposal, which is based on semantic descriptions of non-functional properties and user preferences, is described. The modeling approach taken is to model non-functional properties of services as WSML rules. Furthermore, user preferences and weights are also modeled using rules within goals. These descriptions are processed by a hybrid service ranker, whose architecture is also depicted in this work, along with a run-through using an implemented prototype and the Shipment Discovery scenario from SWS Challenge.

Our approach extends WSMO descriptions in order to express user preferences with utility functions. Moreover, our hybrid architecture allows to perform service ranking tasks using different reasoners and CP solvers, decoupling preferences definition with actual reasoners used.

As future work, we plan to further test the prototype of our hybrid service ranker, using several use case scenarios in order to perform a thoroughly evaluation of it. Additionally, we plan to study and integrate different reasoners and CP solvers, comparing their performance and features. Thus, more complex utility functions have to be tested within our proposal, possibly defining a comprehensive catalog of user preferences definitions.

References

1. J. M. García, D. Ruiz, A. Ruiz-Cortés, O. Martín-Díaz, and M. Resinas. An hybrid, QoS-aware discovery of semantic web services using constraint programming. In B. Krämer, K.-J. Lin, and P. Narasimhan, editors, *ICSOC 2007*, volume 4749 of *LNCS*, pages 69–80. Springer, 2007.
2. J. M. García, I. Toma, D. Ruiz, A. Ruiz-Cortés, Y. Ding, and J. M. Gómez. Ranking semantic web services using rules evaluation and constraint programming. In *JSWEB 2008*, pages 111–119, 2008. To appear.

3. J. González-Castillo, D. Trastour, and C. Bartolini. Description logics for matchmaking of services. Technical Report HPL-2001-265, Hewlett Packard Labs, 2001.
4. K. Kritikos and D. Plexousakis. Semantic QoS metric matching. In *ECOWS 2006*, pages 265–274. IEEE Computer Society, 2006.
5. L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Int. World Wide Web Conference*, pages 331–339, 2003.
6. C. Lutz and U. Sattler. A proposal for describing services with DLs. In *Int. Workshop on Description Logics*, 2002.
7. J. Pathak, N. Koul, D. Caragea, and V. G. Honavar. A framework for semantic web services discovery. In *WIDM '05: Proceedings of the 7th annual ACM international workshop on Web information and data management*, pages 45–50, New York, NY, USA, 2005. ACM Press.
8. D. Roman, H. Lausen, and U. Keller (Ed.). Web service modeling ontology (WSMO). Working Draft D2v1.4, WSMO, 2007. Available from http://www.wsmo.org/TR/d2/v1.4/.
9. A. Ruiz-Cortés, O. Martín-Díaz, A. Durán-Toro, and M. Toro. Improving the automatic procurement of web services using constraint programming. *Int. J. Cooperative Inf. Syst*, 14(4):439–468, 2005.
10. H. Lausen S. Grimm, U. Keller and G. Nagypal. A reasoning framework for rule-based WSML. In *In Proceedings of 4th European Semantic Web Conference (ESWC) 2007*. IEEE Computer Society, 2007.
11. N. Steinmetz and I. Toma (Ed.). The Web Service Modeling Language WSML. Technical report, WSML, 2008. WSML Working Draft D16.1v0.3. http://www.wsmo.org/TR/d16/d16.1/v0.3/.
12. K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated discovery, interaction and composition of semantic web services. *J. Web Sem.*, 1(1):27–46, 2003.
13. I. Toma and D. Foxvog. Non-functional properties in web services. Working Draft D28.4v0.1, Digital Enterprise Research Institute (DERI), August 2006. Available from http://www.wsmo.org/TR/d28/d28.4/v0.1/.
14. I. Toma, D. Roman, D. Fensel, B. Sapkota, and J. M. Gomez. A multi-criteria service ranking approach based on non-functional properties rules evaluation. In B. Krämer, K.-J. Lin, and P. Narasimhan, editors, *ICSOC 2007*, volume 4749 of *LNCS*, pages 435–441. Springer, 2007.
15. X. Wang, T. Vitvar, M. Kerrigan, and I. Toma. A QoS-aware selection model for semantic web services. In A. Dan and W. Lamersdorf, editors, *ICSOC 2006*, volume 4294 of *LNCS*, pages 390–401. Springer, 2006.
16. L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, 2004.
17. C. Zhou, L. Chia, and B. Lee. DAML-QoS ontology for web services. In *IEEE International Conference on Web Services*, pages 472–479, 2004.