Ranking Semantic Web Services Using Rules Evaluation and Constraint Programming

José María García[1], Ioan Toma[2], David Ruiz[1], Antonio Ruiz-Cortés[1], Ying Ding[3], and Juan Miguel Gómez[4⋆]

[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
[3] Indiana University
Bloomington, IN 47405, USA
dingying@indiana.edu
[4] Carlos III University
Madrid, Spain
juanmiguel.gomez@uc3m.es

Abstract. Current Semantic Web Services discovery and ranking proposals are based on user preferences descriptions whose expressiveness are limited by the underlying logical formalism used. Thus, highly expressive preference descriptions, such as utility functions, cannot be handled by the kind of reasoners traditionally used to perform Semantic Web Services tasks. In this work, we outline a hybrid approach to allow the introduction of utility functions in user preferences descriptions, where both rules evaluation and constraint programming are used to perform the ranking process. Our proposal extends the Web Service Modeling Ontology with these descriptions, providing a highly expressive framework to specify preferences, and enabling a more general ranking process, which can be performed by different engines.

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 [4, 6, 7, 13], among others. 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.

⋆ 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.

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 the quality of service preferences with respect to parameters that cannot be considered 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 [8, 16, 18], preferences can only be expressed using tendencies or order conditions, so their expressiveness is restricted. Other proposals use utility functions to describe preferences [5, 10], which provide higher expressiveness and make use of Constraint Programming (CP) to perform the ranking. 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)[9]. Thus, user preferences are modeled using the Web Service Modeling Language (WSML)[12], 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, and 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. Related work is discussed in Sec. 4. Finally, Sec. 5 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 description (Sec. 2.1) and user requests and preferences (Sec. 2.2).

2.1 Service Description

In WSML[12], 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. [14]).

For exemplification purposes we use the SWS Challenge[5] Shipment Discovery scenario. We are mainly interested in two aspects of shipment services for this particular

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

scenario, namely discounts 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 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. 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
    nonFunctionalProperty obligations
     definition
      definedBy
        //in case the package is lost or damaged Runners 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 Runners 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. Similarly other non-functional properties can be encoded using WSML rules.

2.2 User Preferences

User preferences express how important certain non-functional parameters 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 parameter, giving information about the preferred range of values for that non-functional parameter. 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).

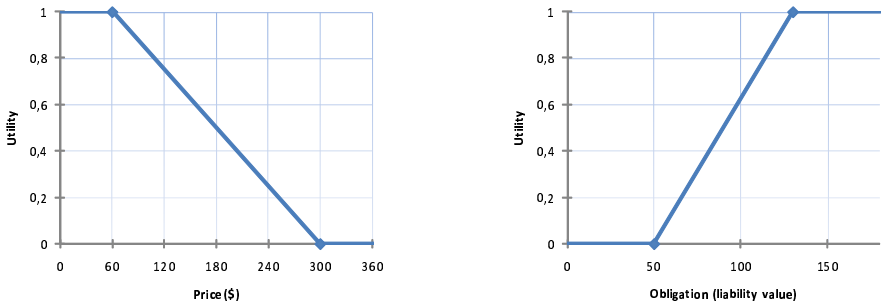

Fig. 1. Price and obligations utility functions.

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 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, so both the requested capability and the user preferences are described using the same formalism. 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.

Listing 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#DefinitionPreference
 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 DefinitionPreferece
  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.

      hasPreference(up#hasObligations, ?value):-
       up#hasObligations[value hasValue ?hasObligationsValue] and ?hasObligationsValue < 130
       and ?hasObligationsValue >= 50 and ?value=(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.

      hasPreference(up#hasPrice, ?value):-
       up#hasPrice[value hasValue ?hasPriceValue] and ?hasPriceValue <= 300
       and ?hasPriceValue > 60 and ?value=(3000-10*?hasPriceValue-500)/24.

axiom DefinitionWeights
  definedBy
      hasWeights(up#hasObligations, 40).
      hasWeights(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 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 kind of utility functions.

To handle WSML logical expressions/axioms we use the IRIS[6] reasoner. In fact, any other reasoner that can handle WSML rules evaluation could be used, e.g. KAON2[7] or MINS[8], provided that it is integrated using the WSML2Reasoner framework [11]. For the CP part the Choco[9] system is used. The overall envision architecture is provided in Fig. 2.

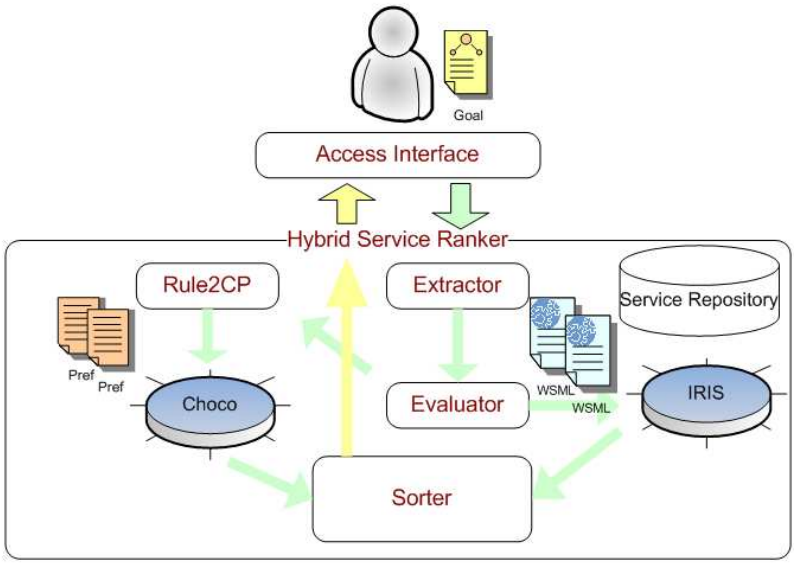

Fig. 2. Hybrid Architecture for Service Ranking.

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

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

goal through an *Access interface* component. That request is formalized as presented in Sec. 2. Moreover, that access interface can be used within different discovery solutions (such as UDDI [1] or web service search engines like seekda[10]), provided that services are annotated with WSML.

Once submitted into 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 (the importance the user gives to each non-functional property).

For the evaluation of the WSML rules that are used to encode the non-functional properties of the services 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 overall process is based on our previous work described in [15]. 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 rank values for each service are ordered and the ranked list of services are provided back 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.

4 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 depending on matching degrees and weighted functions [8]. Similarly, Zhou *et al.* rank services using matching degrees and provide an extension to DAML-S in order to include quality-of-service profiles [18]. 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 [16]. Another WSMO extension, which our proposal is based on, is proposed in [15], where a multi-criteria ranking approach is presented.

Although not specifically focused on ranking services, [17] 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 [10]. In that work, Ruiz-Cortés *et al.* perform the ranking using CP. This paradigm is also used in [5], where an ontology of non-functional properties is introduced to define

[10] http://seekda.com/

them, though user preferences are not semantically defined. In [2] a generic hybrid architecture to perform discovery, ranking and selection is proposed, which is extended in [3] with the inclusion of a first approach to semantically describe user preferences.

5 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. 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 the preferences definition with actual reasoners used.

As future work, we plan to further develop a prototype of our hybrid service ranker, so a full set of test cases can be performed using our ranking approach. 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. L. Clement, A. Hately, C. von Riegen, and T. Rogers. UDDI Version 3.0.2. Technical report, OASIS, October 2004.
2. 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.
3. J. M. García, D. Ruiz, A. Ruiz-Cortés, and M. Resinas. Semantic Discovery and Selection: A QoS-Aware, Hybrid Model. In *The 2008 International Conference on Semantic Web and Web Services*. CSREA Press, 2008.
4. J. González-Castillo, D. Trastour, and C. Bartolini. Description logics for matchmaking of services. Technical Report HPL-2001-265, Hewlett Packard Labs, 2001.
5. K. Kritikos and D. Plexousakis. Semantic QoS metric matching. In *ECOWS 2006*, pages 265–274. IEEE Computer Society, 2006.
6. 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.
7. C. Lutz and U. Sattler. A proposal for describing services with DLs. In *Int. Workshop on Description Logics*, 2002.
8. 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.
9. 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/.

10. 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.
11. 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.
12. 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/.
13. 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.
14. 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/.
15. 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.
16. 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.
17. 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.
18. C. Zhou, L. Chia, and B. Lee. DAML-QoS ontology for web services. In *IEEE International Conference on Web Services*, pages 472–479, 2004.