Automated Selection of Configurable Web Services

Steffen Lamparter, Anupriya Ankolekar

Institute AIFB
Universität Karlsruhe (TH)
Germany
{sla,aan}@aifb.uni-karlsruhe.de

Abstract

To bring service-oriented architectures to their full potential, automatic service discovery and selection mechanisms are required. In this paper, a service selection component is presented that supports offers providing multiple configurations of a service. The selection algorithm ranks the offered services and their configurations according to the requester's preferences and thus facilitates personalized selection strategies. In addition, the approach leverages existing Web standards to provide a maximal degree of interoperability between service providers and their customers leading to significant efficiency gains. The approach is implemented prototypically and the performance is evaluated by means of a simulation.

1 Introduction

Service-oriented architectures (SOA) as a paradigm where applications are built by composing loosely coupled, highly interoperable, reusable services have become increasingly popular in recent years. This flexible style of implementing applications promises cost savings in software development and faster adaption of business processes to changing environments. Realizing such a flexibility requires automatic as well as efficient Web service discovery and selection mechanisms. These mechanisms depend heavily on the way services are described. Approaches that are solely based on WSDL-descriptions are inadequate since XML and XMLSchema do not provide sufficient expressivity to create and relate rich datatypes [MM03]. Therefore, these mechanisms are not amenable to a high degree of automation. In order to tackle this problem several mechanisms have been proposed that depend on much richer service descriptions. Most of them describe services using formal ontologies (e.g. WSMO [KLP+04], OWL-S [SPAS03], WSDL-S [AFJ+05]). However, none of these approaches address the fact that Web services are highly configurable products that can be offered by multiple parties (providers) with different attributes and under different conditions. This enables a high degree of product differentiation allowing providers to customize their services according to the specific needs of their customers. Clearly, this improves the utility of the transaction for both participants. Consider, for example, a route planning web service, which offers the service of computing a road route between

two locations. Various configurations of the service may take into account the current traffic situation or weather situation when computing the route, or the service may be configured to compute the shortest or quickest route, one that avoids small roads and so on. Naturally, each configuration may have a different price attached. Decision making in markets with such complex services generally requires that both seller pricing functions as well as buyer scoring (preference) functions be taken into account.

In [LAO+06] a general policy framework is described that can be used to express pricing and scoring functions. The framework relies on existing or emerging internet standards and thus provides a high degree of interoperability. In this paper, we extend the work in several directions. First, it is shown how the policy framework can be applied to describe configurable Web service offers and requests. Second, the framework is augmented with an abstract selection model, which is independent from specific formalisms and implementations. Third, we present an approach to rank offers in scenarios where both requests and offers are configurable. In this context, we also show how preferences can be expressed directly within a query. The selection algorithm in the previous work deals only either with configurable requests or configurable offers, which are both stored in the knowledge base. Finally, we present a concrete implementation and initial performance evaluations of the system.

The paper is structured as follows. In Section 2 an abstract selection model is presented. In Section 3 this model is formalized with the standardized Web languages OWL, SWRL and SPARQL in order to provide interoperablity between Web service requesters and providers. After presenting an implementation and evaluation in Section 4, we discuss related work in Section 5 and conclude the work with a short outlook in Section 6.

2 Abstract Selection Model

First, a *selection* is defined as a decision for the *best* available alternative, i.e. the Web service that is most appropriate to fulfill a certain task. In general, decisions require a choice and criteria by which different choices are judged. Hence, a selection can be regarded as an optimization problem with a certain objective function resulting in an understanding of "better" and "worse". In this section, we first introduce an abstract notation of the fundamental concepts in the domain and then present the optimization problem that has to be solved in order to derive a preference structure over the service offers.

Definition 1 (Web Service Configuration) *Let $\mathcal{S} = \{s_1, \ldots, s_{|\mathcal{S}|}\}$ be the set of all Web services. Then, a Web service $s_i \in \mathcal{S}$ is described by the tuple $s_i = (\mathcal{I}_i, \mathcal{O}_i, \mathcal{C}_i)$, where $\mathcal{I}_i$ represent the set of input messages that are required by the service s_i and $\mathcal{O}_i$ the set of output messages that are returned by the service s_i, respectively. Furthermore, a Web service is characterized by a set of feasible configurations $\mathcal{C}_i$ with $\mathcal{C}_i \subseteq \mathcal{C}$, where $\mathcal{C}$ is the cartesian product of the discrete*

service attributes $\mathcal{A}_1, \ldots, \mathcal{A}_n$*, i.e.* $C = \prod_{r \in \{1,\ldots,n\}} A_r$*. In this context, each* $c \in C$ *is a vector* $c = (a_{1k_1}, \ldots, a_{jk_j}, \ldots, a_{nk_n})$*, where* a_{jk_j} *represents the* k^{th} *value of attribute* j*.*

Recall the route planing example mentioned in section 1. Here the set $\mathcal{I}_i$ represents locations and O_i the route between these locations. Moreover, we have three attributes: (i) The route type which refers either to the quickest or cheapest route; (ii) one attribute that indicates whether traffic information is taken into account; (iii) and one attribute that allows to include or exclude weather information. These attributes are used to configure the service according to the requesters preferences. For instance, one specific route service configuration is a routing functionality that returns the quickest route including traffic and weather information. Note that according to Definition 1, service functionality is solely defined by inputs as well as outputs and attributes comprise only discrete and static non-functional service properties.
In order to enable Web service transactions between providers and customers, Web service offers and requests have to be specified. Their main purpose is to attach prices as well as preferences to the service configurations. In the following, we formally define how offers and requests can be specified for configurable services. More general approach (e.g. including discount rules) see [BK05].

Definition 2 (Web Service Offer) *Assume a set of providers* $\mathcal{P}$ *as well as a set of Web service offers* Θ*. Then,* $p \in \mathcal{P}$ *is an arbitrary provider offering service* $o_p \in \Theta$*. A Web service offer is characterized by a vector* $o_p = (s_p, G_p)$*, where* $s_p \in \mathcal{S}$ *represents the provided service and* $G_p : C_p \rightarrow \mathbb{R}$ *the pricing function that assigns a certain price to each configuration* $c \in C_p$ *of the service* s_p*. We assume that the pricing function is described by an additive function, where* g_{pj} *represents the pricing function of provider* p *for attribute* j*.* w_j^g *can be used to adjust the influence of the different attributes on the price.*

$$G_p(c) = G_p(a_{1k_1}, \ldots, a_{jk_j}, \ldots, a_{nk_n}) = \sum_{j=1}^{n} w_j^g g_{pj}(a_{jk_j}) \text{ with } \sum_{j=1}^{n} w_j^g = 1 \qquad (1)$$

That means an offer assigns an additive pricing function to a Web service description. The pricing function maps the configurations contained in the service description to a certain price. This approach allows for encoding pricing information in an efficient way. This is required since adding price markup to each configuration would exhibit combinatorial features [BK05]. By means of Function 1, the provider has to define only *n* pricing functions instead of adding $\prod_{j=1,\ldots,n} |A_j|$ price markups. For instance, in the simple route planning example, pricing information can be expressed via six attribute value/price-tuples instead of adding price markups to eight configurations. Of course, for more complex services the efficiency gains are much higher. In the following, a Web service request is defined analogously.

Definition 3 (Web Service Request) *Given a set of customers $\mathcal{B}$, a Web service request is defined as a vector $r_b = (s_b, F_b)$, where $b \in \mathcal{B}$ is the issuer of the request and $s_b \in \mathcal{S}$ the requested service. The preferences of the customer are defined by a standard additive scoring function $F_b : C_b \rightarrow \mathbb{R}$ that assigns a certain score to each requested configuration $c \in C_b$. A configuration which is not requested leads to a score of zero. f_{bj} is the scoring function of requester b for attribute j. Attribute values that are forbidden a score of minus infinity is assigned, i.e. $f_{bj} = -\infty$. w_j^f is the relative importance of attribute j.*

$$F_b(c) = \begin{cases} \sum_{j=1}^{n} w_j^f f_{bj}(a_{jk_j}) & \text{if } c \in C_b, \\ 0 & \text{otherweise.} \end{cases} \quad \text{with } \sum_{j=1}^{n} w_j^f = 1 \tag{2}$$

For the additive scoring function we have to assume mutual preferential independency between the attributes [KR76]. However, as discussed in [LAO+06], from a technical perspective we can express preferences over dependent attributes using a higher dimensional function. Of course, specification of preferences is much harder for the requesters in this case.
In order to find the services that are suitable for a certain task, the inputs and outputs of the requested service s_b are compared with those of the offered services. This is done according to the following matching rule.

Definition 4 (Functional Match) *Let r_b be a request for service s_b with the corresponding inputs $\mathcal{I}_b$ and outputs $\mathcal{O}_b$. Analogously, o_p is a Web service offer containing the service s_p with the inputs $\mathcal{I}_p$ and outputs $\mathcal{O}_p$. Then, r_b functionally matches o_p iff $\mathcal{I}_p \subseteq \mathcal{I}_b$ and $\mathcal{O}_b \subseteq \mathcal{O}_p$. A functional match is indicated by the notation $r_b =_f o_p$. The set $\Theta^r \subseteq \Theta$ contains the service offers that functionally match a given request r_b, i.e. $\Theta^r = \{o_b \in \Theta | r_b =_f o_b\}$*

The intuition behind this approach is that only those Web service offers match that provide at least the requested information while requiring at most the input specified in the request.
Finding the optimal service configuration involves two steps: selecting the best configuration for each provider and choosing the best provider based on the optimal configurations. First, we formulate the optimization problem that allows determining the best configuration $c \in C_p$ offered by a provider $p \in \mathcal{P}$.
The objective function that has to be optimized represents the difference between the score assigned to a certain configuration by the requester and the price of this configuration. The corresponding optimization problem can be formulated as follows:

$$\max_{c_i \in C_p} F_b(c_i) - G_p(c_i) \tag{3}$$

A naive approach to solve this problem is to iterate over all configurations, calculate the individual utilities and rank the configurations accordingly. The overall complexity of such an

algorithm is given by $|O| \sum_j |\mathcal{A}_j|$, where $|O|$ is the number of offers and $\sum_j |\mathcal{A}_j|$ the number of possible configurations. Since the number of configurations is exponential in the number of attributes this approach might be inefficient for very complex services. However, in a first step such a naive approach seems to be appropriate, since current Web service descriptions usually consider only relatively few non-functional properties.
Having determined the optimal configuration c_o^* and utility u_o^* for each offer o, we derive a preference structure as follows:

Definition 5 (Preference Structure) *A preference structure on O is defined by the complete, transitive, and reflexive relation $\succeq$, which defines a weak preference order over the alternatives as follows: $\forall o_1, o_2 \in O$: $o_1 \succeq o_2 \Leftrightarrow o_1$ is preferred to o_2. Consequently, a preference structure can be constructed from the requester's utility as follows: $o_1 \succeq o_2 \Leftrightarrow u_{o_1}^* \geq u_{o_2}^*$.*

Therefore, a reasonable selection rule would be to select $o^* \in O$ iff $\forall o \in O : o^* \succeq o$. Of course, depending on the application also other matching approaches might be required. For example, one might want to select the ten best services to invite them for further negotiations.
Based on the abstract selection model introduced above, the next section deals with implementing this model in an open and heterogenous environment using existing standards and tools.

3 Ontology-based Representation

In this section, we show how the abstract selection model introduced above can be implemented with a standardized logical formalism providing a common understanding between providers and customers. We realize this by means of ontologies, which became an important technology for knowledge sharing in distributed, heterogeneous environments, particularly in the context of the *Semantic Web*.[1]

3.1 Ontology formalism

An ontology is a set of logical axioms that formally define a shared vocabulary [Gru93]. By committing to a common ontology, software agents can make assertions or ask queries that are understood by the other agents.
In order to guarantee that these formal definitions are understood by other parties (e.g. in the web), the underlying logic has to be standardized. The Web Ontology Language (OWL) standardized by the World Wide Web Consortium (W3C) is a first effort in this direction [W3C04]. OWL-DL is a decidable fragment of OWL and is based on a family of knowledge representation formalisms called *Description Logics (DL)* [BCM+03]. Consequently, our notion of an ontology is a DL knowledge base expressed via RDF/XML syntax to ensure compatibility with

[1] `http://www.w3.org/2001/sw/`

DL Syntax	Semantics
$\top$	$\Delta^{\mathcal{I}}$
$\bot$	$\emptyset$
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
$\forall R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$
$\exists R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$
$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$

Table 1: Selected DL constructs and their model theoretic semantics (for a full list see [BCM+03]).

existing World Wide Web languages. The meaning of the modeling constructs provided by OWL-DL like concepts, relations, datatypes, individuals and data values is formally defined via a model theoretic semantics. The mapping for certain common DL axioms is shown in table 1. The meaning of an axiom defines certain constraints on the model. For example, we can define that the concept *Book* is a subconcept of *Product* (i.e. $Book \sqsubseteq Product$). In this case, the interpretation of *Book* has to be a subset of the interpretation of *Product*, i.e. the set of objects that are books is a subset of the set of objects that are products ($Book^{\mathcal{I}} \subseteq Product^{\mathcal{I}}$). By means of the interpretation $\mathcal{I}$ the model introduced in section 2 can be expressed using Description Logics in a straightforward way as shown in the next sections.

However, we require additional modeling primitives not provided by OWL-DL. For example, modeling triangle relations between concepts is required. In contrast to OWL, rule languages can be used to express such triangle relation. The Semantic Web Rule Language (SWRL) [HPS04, HPSB+04] allows us to combine rule approaches with OWL. Since reasoning with knowledge bases that contain arbitrary SWRL expression usually becomes undecidable [HPS04], we restrict ourself to *DL-safe* rules [MSS05]. DL-safe rules keep the reasoning decidable by placing constraints on the format of the rule, namely each variable occurring in the rule must also occur in a non-DL-atom in the body of the rule. This means the identity of all objects referred to in the rule has to be known explicitly. Since we deal only with known instances in our application, this is no restriction to our apporach. To query and reason over a knowledge base containing OWL-DL as well as DL-safe SWRL axioms we use the KAON2 inference engine[2].

For the reader's convenience we define DL axioms informally via UML class diagrams, where UML classes correspond to OWL concepts, UML associations to object properties, UML inheritance to subconcept-relations and UML attributes to OWL datatype properties [BVEL04]. For representing rules we rely on the standard rule syntax as done in [HPSB+04, MSS05].

[2] available at `http://kaon2.semanticweb.org/`

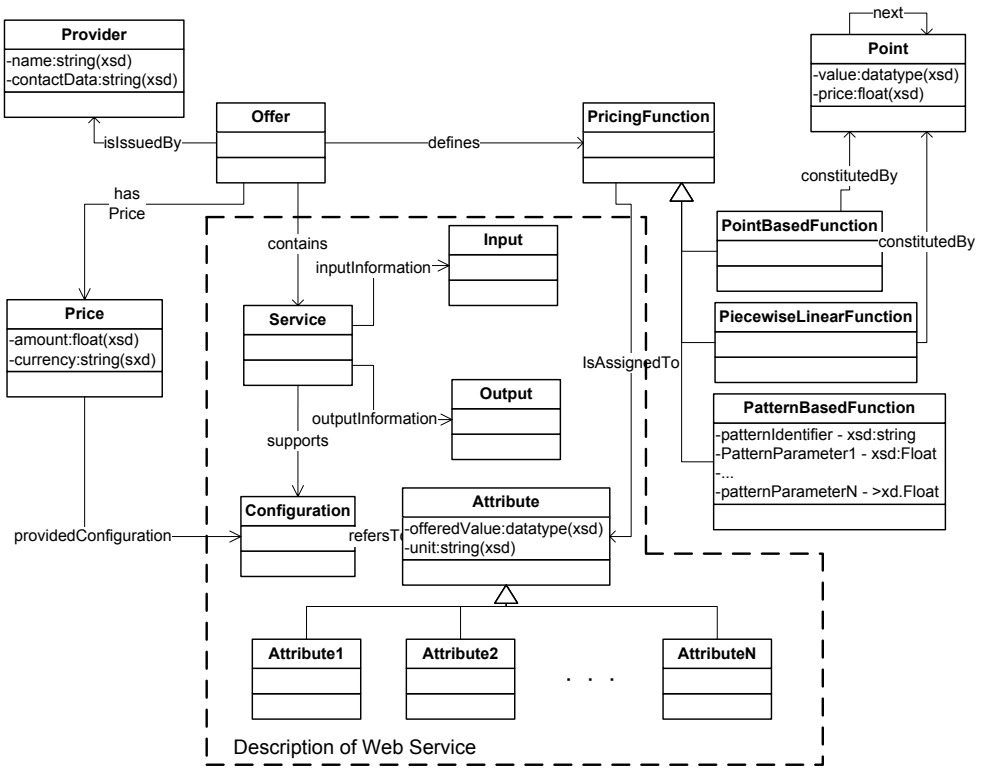

Figure 1: Web Service Offer Ontology

3.2 Modeling Web Services

In this section, it is shown how the abstract Web service model specified in Definition 1 can be formalized using OWL-DL. Note that some additional assumptions are required to derive a formally sound model.

Figure 1 sketches the modeling approach. For modeling Web services the classes within the dashed box are relevant. We introduce the concepts *Service*, *Input*, *Output*, *Configuration* and *Attribute* referring to the sets $\mathcal{S}$, $\mathcal{I}$, $\mathcal{O}$ and $\mathcal{A}_1$ to $\mathcal{A}_n$ in Definition 1, respectively. A *Service* is characterized by their *Inputs*, *Outputs* and *Configurations* involved. This is reflected by the relations *inputInformation*, *outputInformation*, and *supports*, which are formally defined by the following axiom:

$$\begin{aligned} \mathit{Service} \quad \sqsubseteq \quad & \top \sqcap \exists \mathit{inputInformation}.\mathit{Input} \sqcap \forall \mathit{inputInformation}.\mathit{Input} \sqcap \\ & \exists \mathit{outputInformation}.\mathit{Output} \sqcap \forall \mathit{outputInformation}.\mathit{Output} \sqcap \\ & \exists \mathit{supports}.\mathit{Configuration} \end{aligned} \tag{4}$$

This axiom makes sure that each service has at least an *Input* and *Output*. In addition, each service has to support at least one *Configuration*. Recall in our abstract model a configuration $c_i \in C$ is a vector $c_i = (a_{11}, \ldots, a_{ij}, \ldots, a_{nm})$ containing one value a_{ij} for each attribute. Hence, we introduce *Configuration* as those individuals that refer to exactly one individual of each

attribute. Assume the concepts $Attribute1 \sqsubseteq Attribute, \ldots, AttributeN \sqsubseteq Attribute$ which refer to the sets $\mathcal{A}_1$ to $\mathcal{A}_n$ Then a *Configuration* is defined as follows:

$$Configuration \quad \sqsubseteq \quad \top \sqcap =_1 refersTo.Attribute1 \sqcap \cdots \sqcap =_1 refersTo.AttributeN \tag{5}$$

Each *Attribute* has to have at least two datatype properties: one property representing the *offered value* as well as one property defining the *unit* a certain attribute is measured.
This means in the route planning scenario a service would be described by the three object properties *start*, *destination* and *result*, where *start* and *destination* are subproperties of *inputInformation* and *result* is a subproperty of *outputInformation*. [3] Then a route planing service is defined as follows:

$$RoutePlanning \sqsubseteq Service \sqcap =_1 start.Location \sqcap =_1 destination.Location \sqcap \exists result.Route \tag{6}$$

Furthermore, the service is described by three *Attributes*: $Weather \sqsubseteq Attribute$, $Traffic \sqsubseteq Attribute$ and $RouteType \sqsubseteq Attribute$. Each of them could be instantiated by two attribute values, which leads to eight possible configurations that can be provided.
After introducing the primitives for modeling Web services in the next sections we show how offers and requests for a certain Web service can be formalized.

3.3 Offer Specification

According to Definition 2 an offer $o \in \Theta$ is characterized by a provider $p \in \mathcal{P}$ offering a service $s \in \mathcal{S}$ under a certain pricing policy $G_p(c_i)$, where c_i represents an arbitrary service configuration $c_i \in C_p$. For modeling the set of offers Θ we introduce the concept *Offer* which *contains* a *Service* and *is issued by* a *Provider*. These simple facts are visualized in Figure 1. Furthermore, prices are attached to the various Web service configurations by means of a function $G_p(c_i)$. To capture this relations the concept *Price* is introduced that relates an *Offer*, a *Configuration* and the corresponding *amount*. This is required since OWL does not support tertiary relations. The datatype property *amount* is determined by function 1. Such functions are modeled using the concept *Pricing Function*. A thorough discussion how such policies can be modeled using OWL-DL and DL-safe SWRL rules is given in [LAO+06]. In the following we focus one of the approaches - called *Point Based Function*.
In case of a *Point Based Function* the pricing function is modeled by specifying sets of points in $\mathbb{R}^2$ that explicitly map attribute values to prices. This is particularly relevant for nominal attributes. As depicted in Figure 1, *Point Based Functions* are *Pricing Functions* that are *constituted by* a set of *Points*. Thus, the datatype property *value* refers to exactly one attribute

[3] Note that the examples given in this paper are simplified. To be precise one would have to model *Input* and *Output* as roles that are played by information objects representing location or route information. For a more detailed modeling approach based on the foundational ontology DOLCE see [LAO+06, OLG+06]

value a_{jk_j} and the datatype property *price* to exactly one price $g_j(a_{jk_j})$ that is assigned to this attribute value. OWL datatypes mainly rely on the non-list XML Schema datatypes. Depending on the attribute, *value* either points to a *xsd:string*, *xsd:integer* or *xsd:float*. A *price* is represented by a *xsd:float*. The reference to the attribute j for a function g_j is defined via the *isAssignedTo*-relation. Having this information the overall price $G_p(c_i)$ a provider attaches to a certain configuration c_i can be calculated using formula 1, which can be formalized using the following DL-safe rule:

$$\begin{aligned} amount(?p, ?pr) \leftarrow\ & Offer(?o), hasPrice(?o, ?p), providedConfiguration(?p, ?c), \\ & \bigwedge_{j\in\{1,\ldots,n\}} (Attribute_j(?a_j), refersTo(?c, ?a_j), offeredValue(?a_j, ?av_j), \\ & assignedTo(?f_j, ?a_j), constitutedBy(f_j, p_j), value(?p_j, ?v_j), \\ & equal(?v_j, ?av_j), price(?p_j, ?pr_j)), sum(?pr_1, \ldots, ?pr_n, ?pr) \end{aligned} \quad (7)$$

Note that rule 7 is not generic with respect to the number and types of the attributes used. However, a more general rule definition is not possible since this would require allquantification in rule bodies, which is not possible using SWRL. Since the rule is generic with respect to the offers, it is possible to generate the rule once for a certain service type. Hence, we believe that this is no restriction in practice. To improve readability, the conversion of measurement units that might be required is omitted in rule 7. *Point Based Function* thus allow to decrease the number of required price specifications from $\prod_{j=1}^{n} |\mathcal{A}_j|$ to $\sum_{j=1}^{n} |\mathcal{A}_j|$. Using other function specifications, like *Pattern-based Functions* or *Piecewise Linear Functions*, further improvements are possible.

Having shown how offers can be specified using OWL-DL and DL-safe SWRL rules, in the next section we focus on formalizing request in a way that allows a customer to derive a set of services ordered according to her preferences.

3.4 Request Specification

The aim of this step is to formalize the requester's goals in a way that facilitates the discovery of offers meeting these goals. According to Definition 3, a request defines the properties of an object that is required by a customer in an abstract way, i.e. without referring to a concrete name or identifier. In databases, *queries* are seen as such "intentional" denotations of objects [LL87]. For expressing queries we rely on the emerging standard SPARQL[4], which provides a protocol and query language for RDF and OWL-DL ontologies. SPARQL is supported by the reasoner KAON2. The discovery of suitable offers has to be sound and complete, i.e. all relevant offers are returned while no irrelevant offers are contained in the result set. In addition, a ranking of the results according to the requesters preferences should be provided. In the following,

[4]SPARQL, W3C Candidate Recommendation (6 April 2006), available at `http://www.w3.org/TR/rdf-sparql-query/`

we gradually show how a SPARQL-query is formulated that enables expressing a Web Service Request (Definition 3).

In a first step, we determine the set of suitable services, i.e. those services that provide the right functionality. In this context, we mainly implement the functional matching introduced in Definition 4. A functional match is realized if $\mathcal{I}_p \subseteq \mathcal{I}_b$ and $\mathcal{O}_b \subseteq \mathcal{O}_p$. Consequently, in DL this amounts to checking the entailment of concept subsumption, i.e. $Input_p \sqsubseteq Input_b$ and $Output_b \sqsubseteq Output_p$. This is a very common approach already used for service matching in [PKPS02, NSDM03, LH03]. However, note that this is very restrictive and for some applications better solutions might exist (cf. [GMP04]). The following SPARQL-query formulates the matching rule for our route planning service example.

```
PREFIX  wsm: <http://ontoware.org/emo/1.1/>
SELECT ?offer
 WHERE {
    ?offer contains ?service . ?start rdf:type wsm:Location .
    ?dest rdf:type wsm:Location . ?result rdf:type wsm:Route .
    ?service wsm:start ?start ; wsm:destination ?dest ; wsm:result ?result . }
```

In the second step, we add constraints to this query in order to reduce the number of returned matches. For each attribute $\mathcal{A}_j$ the subset $\bar{\mathcal{A}}_j = \{a_{jk} \in \mathcal{A}_j | f_{bj}(a_{jk}) = -\infty\}$ is determined for which the utility function of a customer is zero. Based on the set $\bar{\mathcal{A}}_j$ SPARQL-filter conditions are automatically added to the query. For example, assume a requester lives in a very busy quarter of London and he thus mandatorily requires a route planner with traffic information. In this case we would add the following filter condition:

```
SELECT ?offer, ?configuration
 WHERE { ...
    ?configuration wsm:refersTo ?traffic .
    ?traffic rdf:type wsm:Traffic ; wsm:offeredValue ?trafficValue .
    FILTER ( ?trafficValue = "yes" ) .
 }...
```

A clause in the filter condition is added for all $a_{jk} \in \bar{\mathcal{A}}_j$ and for all attributes j. Note that from a functional point of view the filter conditions are not required since configurations, which contain at least one attribute that is valued by $-\infty$, are ranked very low and thus are neglected in the selection process. However, by introducing the filter conditions the number of possible configurations that have to be ranked is reduced. This might increase the ranking performance. Finally, to facilitate the selection of a service with the corresponding configuration, we order the set of matches according to the preferences of the user. This means we have to determine which of the configurations $c \in C_p$ offered by a provider $p \in \mathcal{P}$ is most advantageous according to the requester's preferences $F_b(c)$. Technically there are two possibilities to realize this: First, the query above is issued and the results are sent to the client, where a preference-based selection is executed locally. Although this approach avoids revealing the customer's preferences, it may

lead to a high communication overhead, where in the worst case the entire repository has to be transferred to the requester. We thus opt for the second alternative. Here preferences are included as part of the request and the ranking is done by the server. This allows deriving only the best matches from a high number of suitable services in the repository. Methods for adding rich preferences to a query language are well-known in literature [Kie02, LL87, AW00]. Usually such queries are called *preference queries* and are implemented by database built-ins as well as SQL syntax extensions. The KAON2 system enables preference queries by allowing built-in predicates in the SPARQL query. Therefore, the SPARQL syntax is extended by the "EVALUATE" keyword. Again we exemplify the approach using a *Point Based Function* definition. But this time we encode the function as a *String* in the query rather than adding it to the knowledge base (as done for offers in Section 3.3). In order to realize this approach the predicate *pbF* is introduced. *pbF* takes a String representation of the tuples representing the *Point Based Function* and an attribute value. The predicate evaluates the *Point Based Function* and returns the score of the attribute value. The query below illustrates this approach using the three attributes of our route planning example. Note that for simplicity reasons scaling as well as weighting issues are omitted.

```
SELECT ?offer, ?configuration, ?u
 WHERE { ...
    ?offer wsm:hasPrice ?price .
    ?price wsm:providedConfiguration ?configuration .
    ?configuration wsm:refersTo ?traffic ; ?weather ; ?routeType .
    ?traffic rdf:type wsm:Traffic ; wsm:offeredValue ?trafficValue .
    EVALUATE ?val1 := pbF("(yes,1),(no,0.3)",?trafficValue).
    ?weather rdf:type wsm:Weather ; wsm:offeredValue ?weatherValue .
    EVALUATE ?val1 := pbF("(yes,0.8),(no,0.6)",?wheaterValue).
    ?routeType rdf:type wsm:RouteType ; wsm:offeredValue ?routeTypeValue .
    EVALUATE ?val1 := pbF("(quickest,0.5),(cheapest,0.5)",?routeTypeValue).
    EVALUATE ?val := sum(?val1,?val2,?val3) .
    EVALUATE ?u := sub(?val,?price) }
 ORDER BY DESC(?u)
```

The result of this query is a set of matches ordered according to the preference structure introduced in Definition 5. Thus, the corresponding selection rules can be applied.
In the next section, we present a concrete implementation of our selection algorithm. In addition, we discuss the performance of the algorithm.

4 Implementation

The algorithm presented in this paper is implemented within a larger framework consisting of two components: A server component provides a repository for Web service offers. The repository is a DL knowledge base that can be queried using the KAON2 reasoner. KAON2 is

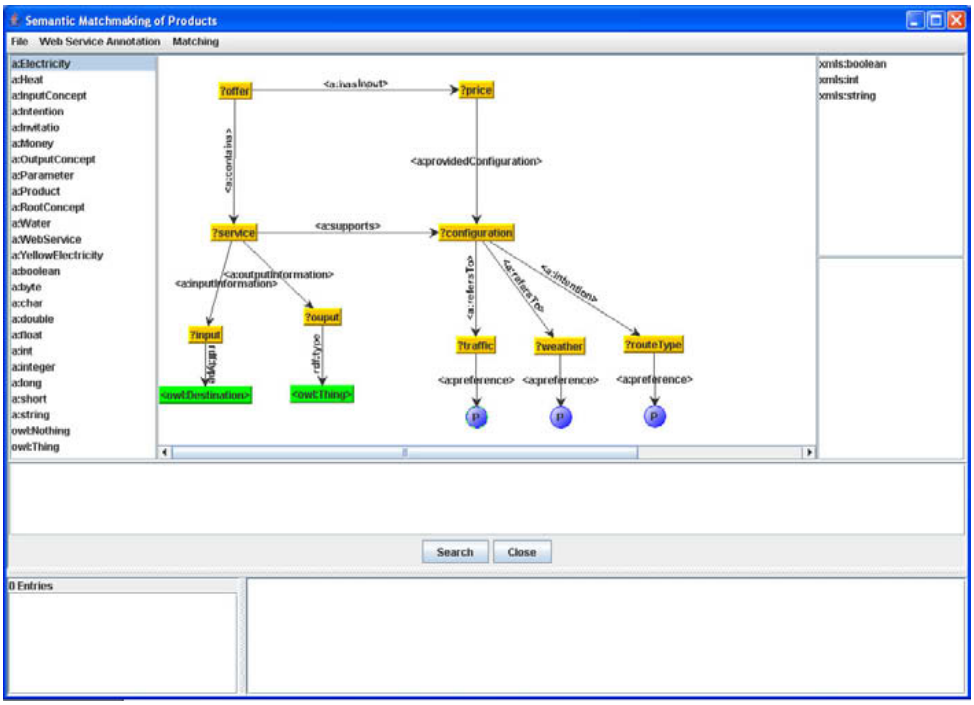

Figure 2: Generation of SPARQL-query.

chosen because it supports the logical fragment required for our offer and request descriptions, while being optimized for query answering [MS06], which is the main focus of the repository. In addition, there are components that transform WSDL and HTML forms to ontology-based descriptions and a Web crawler, which searches the Web for available service descriptions. The second component is a client that facilitates the specification of Web service offers and requests. Since the terminology used by participants might be different, mapping between ontologies can be specified using the formalism presented in [HM05]. Generally, the framework supports more expressive service descriptions than we use in this paper. For example, the service description could include behavioral aspects as presented by Agarwal and Studer [AS06].

Figure 2 shows the request generator which is part of the client tool. It can be used to graphically compose a query. Preferences can be added to all attributes in the query (visualized by a blue circle with a 'P'). By pressing this symbol a window pops up that allows for expressing *Point Based Functions, Piecewise Linear Functions* and *Pattern Based Functions*. Once the search button is pressed, the query is formalized using SPARQL and sent to the server. Ranked results are shown and can be browsed using the windows below the search button.

In order to evaluate the performance of the algorithm we conducted a simulation. In this context, offers and requests are randomly generated using a uniform distribution. The set of offers is stored in the knowledge base and the requests are used to generate the queries. The time between sending the query and receiving the result is measured. In order to avoid possible network delays the simulation is done on a single machine. Several simulation runs are conducted with varying

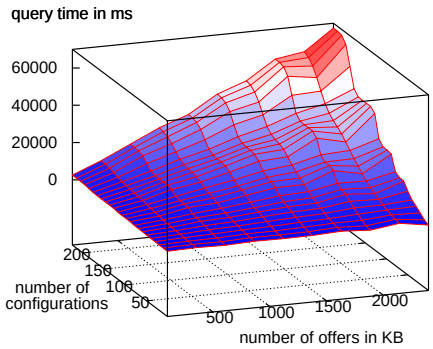

Figure 3: Query time.

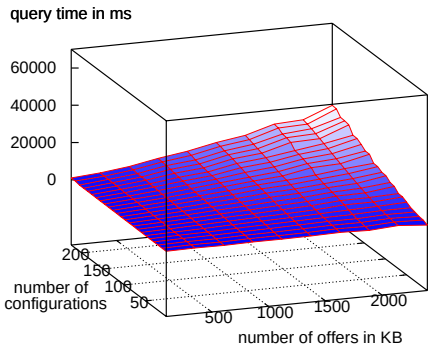

Figure 4: Query time using caching.

number of offers in the knowledge base and varying service complexity. For each setting the average query time is determined based on ten simulation runs. We analyzed the worse case scenario, where all offers functionally match the request, no hard constraints are defined, and all offers provide all configurations. We evaluated two versions of the algorithm:

Algorithm 1: In the first algorithm we simply query the knowledge base and calculate the price of an offer using Rule 7. This has to be done for every relevant configuration, i.e. $|\Theta||C|$ configurations in the worst case.

Algorithm 2: To avoid unnecessary repetition of the same evaluations we introduce *caching* of prices in algorithm 2. After calculating the price for a certain configuration according to the pricing policies in the offer, we store this price as an instance in the knowledge base which avoids additional evaluations for further requests. However, this approach increases the size of the knowledge base by additional $5|\Theta||C| - 4|\Theta|$ axioms compared to Algorithm 1. In a first step, we do not limit the cache size. However, in case of limited storage/memory capacity an adequate caching replacement strategy has to be introduced.

Figure 3 and 4 show the interdependency between the number of offers, the number of configurations in an offer, and the query time for algorithm 1 and 2, respectively. For scenarios with a low number of orders (< 500) or a low number of configurations (< 100) query answering is realized under five second. However, query answering slows down to 70 seconds with 2500 orders and 250 configurations in the knowledge base, which is mainly due to the high number of price calculations. Therefore, the caching of prices used in Algorithm 2 speeds up service selection considerably. However, one should be aware that Algorithm 2 is significantly more resource demanding.

Generally, the evaluation shows that our selection algorithm might be applicable in scenarios where either the number of offers or the number of configurations per offer is moderate or where the selection performance is not crucial. To improve the performance introducing an instance for each configuration should be avoided and the optimization should be done directly based on the pricing functions, possibly with a built-in predicate implementing a simplex algorithm.

Moreover, we could exploit the additive structure of pricing and scoring functions, perform the maximization per attribute and aggregate the maximal values for all attributes. Thereby, the complexity of the problem can be considerably reduced.

5 Related Work

First approaches addressing the configurability of services are policy languages like WS-Policy, EPAL and WSPL. They allow to define which configuration are supported by a provider or desired by a requestor, but no prices or preferences can be attached. WS-Agreement [Gri05] extends WS-Policy in this direction and provides the means for attaching prices and preferences to configurations. However, all these specifications are based purely on XML and thus lack formal semantics. KAoS [UBJ04] and REI [Kag04] are ontology-based approach for expressing policies. However, they also evaluate either to true or false and thus provide no ranking of the alternatives.

In literature we can identify three major branches of work that strive for ranking of suitable services. First, there are logic-based approaches (e.g., [PKPS02], [GMP04], [NSDM03]) that allow for different degrees of matches based on partial or incomplete matches. However, such rankings are typically rather coarse and one can argue that pure logical matchmaking without value reasoning is not sufficient [SRT05, KFKS05]. Second, there are matchmaking approaches purely based on information retrieval techniques like [BK06]. Here rankings are calculated by defining similarity measures for service properties. Klusch and colleagues [KFKS05] extend logic-based matchmaking with syntactic measures. This is similar to our approach, since we use a logic-based approach for matchmaking of service functionality. In addition, similarity measures can be seen as special preference functions. However, our selection approach is not limited to similarity-based preferences. It is also possible to specify the valuation of certain alternatives explicitly. This is also possible in the system presented by Balke and Wagner [BW03]. They use SQL-based preference queries introduced in [Kie02]. However, their work is not based on semantic service annotations. A third branch of work relies on logical rule languages to specify preferences (as in our case SWRL). Prominent examples are SweetDeal [GP03] and the work by Oldham and colleagues [OVSH06]. Although these approaches show how preferences can be formally represented, they currently lack a formal selection model as we presented in section 2.

All approaches above consider only the case of selecting a single service. Approaches beyond this simple case are presented in [SNVW06] and [ZBN+04]. [SNVW06] presents an multi-attribute combinatorial auction for grid service markets with multiple sellers and buyers bidding on complex service bundles. Zeng and colleagues [ZBN+04] present two methods for selecting a service based on several QoS-criteria. In this context, they consider not only one single service, but optimize the utility for an entire composition of services using integer programming.

However, the problem of service selection for multiple service requests is beyond the scope of this paper.

6 Conclusion

The work in this paper addresses a problem that arises with the increasing differentiation of services in electronic markets. This requires mechanisms to describe configurable services and algorithms to discover and select suitable services and configurations. It is shown, how the abstract selection model can be implemented by leveraging standardized web languages, such as OWL-DL, SWRL and SPARQL. This facilitates interoperability in heterogenous and open environments. In a first step, a rather simple optimization algorithm is applied to rank the service offers according to the preferences of a requester. The algorithm is evaluated by means of a simulation. The evaluations indicate that, while being suitable for rather simple service descriptions or small repositories, the system is not yet capable of selecting from a large repository of complex services in run-time. We are currently working on the scalability of the system and plan to add a built-in predicate which efficiently implements the optimization problem. In doing this, iterating over all configurations can be avoided and the number of instances in the repository can be reduced. Moreover, we plan to extend our approach in order to allow specifying preferences also for functional, dynamic and continuous service properties.

Acknowledgments

This work was funded by the German Research Foundation (DFG) in scope of Graduate School Information Management and Market Engineering.

References

[AFJ+05] R. Akkiraju, J. Farrell, J.Miller, M. Nagarajan, M. Schmidt, A. Sheth, and K. Verma. Web Service Semantics - WSDL-S, " A joint UGA-IBM Technical Note, version 1.0,. Technical report, April 2005.

[AS06] S. Agarwal and R. Studer. Automatic matchmaking of web services. In *Int. Conf. on Web Services (ICWS'06)*, Chicago, USA, 2006. IEEE Computer Society.

[AW00] R. Agrawal and E. L. Wimmers. A framework for expressing and combining preferences. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 297–306, New York, NY, USA, 2000. ACM Press.

[BCM+03] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory Implemenation and Applications*. Cambridge University Press, 2003.

[BK05] M. Bichler and J. Kalagnanam. Configurable offers and winner determination in multi-attribute auctions. *European Journal of Operational Research*, 160(2):380–394, January 2005.

[BK06] A. Bernstein and C. Kiefer. Imprecise RDQL: Towards Generic Retrieval in Ontologies Using Similarity Joins. In *21th Annual ACM Symposium on Applied Computing (SAC)*, New York, NY, USA, 2006. ACM Press.

[BVEL04] S. Brockmans, R. Volz, A. Eberhart, and P. Löffler. Visual modeling of OWL DL ontologies using UML. In S. M. et al., editor, *Proc. of the 3rd International Semantic Web Conference*, pages 198–213, Hiroshima, Japan, November 2004. Springer LNCS.

[BW03] W.-T. Balke and M. Wagner. Towards personalized selection of web services. In *Proc. of the 12th Int. World Wide Web Conference*, 2003.

[GMP04] S. Grimm, B. Motik, and C. Preist. Variance in e-business service discovery. In *Semantic Web Services: Preparing to Meet the World of Business Applications, workshop at ISWC 2004*, 2004.

[GP03] B. Grosof and T. Poon. SweetDeal: Representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In *Proc. of the 12th Int. Conf. on the World Wide Web (WWW 2003)*, Budapest, Hungary, May 2003.

[Gri05] Grid Resource Allocation and Agreement Protocol Working Group. Web services agreement specification. `https://forge.gridforum.org/projects/graap-wg/document/WS-AgreementSpecification/en/7`, June 2005.

[Gru93] T. R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993.

[HM05] P. Haase and B. Motik. A mapping system for the integration of owl-dl ontologies. In A. Hahn, S. Abels, and L. Haak, editors, *IHIS 05: Proceedings of the first international workshop on Interoperability of heterogeneous information systems*, pages 9–16, 2005.

[HPS04] I. Horrocks and P. F. Patel-Schneider. A proposal for an OWL rules language. In *Proceedings of the 13th International Conference on the World Wide Web (WWW 2004)*, pages 723–731, New York, USA, 2004. ACM Press.

[HPSB+04] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A semantic web rule language combining OWL and RuleML. W3C Submission, avilable at `http://www.w3.org/Submission/SWRL`, May 2004.

[Kag04] L. Kagal. *A Policy-Based Approach to Governing Autonomous Behavior in Distributed Environments*. PhD thesis, University of Maryland Baltimore County, Baltimore MD 21250, November 2004.

[KFKS05] M. Klusch, B. Fries, M. Khalid, and K. Sycara. Owls-mx: Hybrid semantic web service retrieval. In *Proceedings of 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web*, Arlington VA, USA, 2005. AAAI Press.

[Kie02] W. Kießling. Foundations of preferences in database systems. In *VLDB'02: Proc. of the 13th Int. Conf. on Very Large Databases*, pages 311–322, 2002.

[KLP+04] U. Keller, R. Lara, A. Pollers, I. Toma, M. Kifer, and D. Fensel. WSMO Web Service Discovery, November 2004. http://www.wsmo.org/TR/d5/d5.1/v0.1.

[KR76] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. J. Wiley, New York, 1976.

[LAO+06] S. Lamparter, A. Ankolekar, D. Oberle, R. Studer, and C. Weinhardt. A policy framework for trading configurable goods and services in open electronic markets. In *Proceedings of the 8th Int. Conference on Electronic Commerce (ICEC'06)*, New Brunswick, Fredericton, Canada, August 2006.

[LH03] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *WWW '03: Proceedings of the twelfth international conference on World Wide Web*, pages 331–339. ACM Press, 2003.

[LL87] M. Lacroix and P. Lavency. Preferences; putting more knowledge into queries. In *VLDB '87: Proceedings of the 13th International Conference on Very Large Data Bases*, pages 217–225, San Francisco, CA, USA, 1987.

[MM03] D. J. Mandell and S. McIlraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. In D. Fensel, K. P. Sycara, and J. Mylopoulos, editors, *2nd Int. Semantic Web Conference (ISWC)*, volume 2870 of *LNCS*, pages 227–247, Sanibel Island, FL, USA, 2003. Springer.

[MS06] B. Motik and U. Sattler. A comparison of reasoning techniques for querying large description logic aboxes. In *Proc. of the 13th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2006)*, Phnom Penh, Cambodia, November 2006.

[MSS05] B. Motik, U. Sattler, and R. Studer. Query answering for OWL-DL with rules. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60, JUL 2005.

[NSDM03] T. D. Noia, E. D. Sciascio, F. M. Donini, and M. Mongiello. A system for principled matchmaking in an electronic marketplace. In *WWW '03: Proceedings of the twelfth international conference on World Wide Web*, pages 321–330. ACM Press, 2003.

[OLG+06] D. Oberle, S. Lamparter, S. Grimm, D. Vrandecic, S. Staab, and A. Gangemi. Towards ontologies for formalizing modularization and communication in large software systems. *Journal of Applied Ontology*, 1(2), 2006.

[OVSH06] N. Oldham, K. Verma, A. Sheth, and F. Hakimpour. Semantic WS-Agreement partner selection. In *Proc. of the 15th Int. WWW Conference*, Edinburgh,UK, 2006.

[PKPS02] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara. Semantic matching of web services capabilities. In *1st International Semantic Web Conference (ISWC)*, LNCS 2342, pages 333–347, 2002.

[SNVW06] B. Schnizler, D. Neumann, D. Veit, and C. Weinhardt. Trading grid services - a multi-attribute combinatorial approach. *Forthcoming in European Journal of Operational Research*, 2006.

[SPAS03] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated discovery, interaction and composition of semantic web services. *Journal of Web Semantics*, 1(1), 2003.

[SRT05] A. P. Sheth, C. Ramakrishnan, and C. Thomas. Semantics for the semantic web: The implicit, the formal and the powerful. *Int. J. Semantic Web Inf. Syst.*, 1(1):1–18, 2005.

[UBJ04] A. Uszok, J. M. Bradshaw, and R. Jeffers. KAoS: A Policy and Domain Services Framework for Grid Computing and Semantic Web Services. In *Trust Management: 2.d Int. Conference, iTrust 2004, Oxford, UK*, volume 2995 of *LNCS*, pages 16–26. Springer, 2004.

[W3C04] W3C. Web ontology language (OWL). `http://www.w3.org/2004/OWL/`, 2004. W3C Recommendation.

[ZBN+04] L. Zeng, B. Benatallah, A. 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.