Preference-based Selection of Highly Configurable Web Services

Steffen Lamparter, Anupriya Ankolekar, Rudi Studer
Institute AIFB, University of Karlsruhe (TH)
76128 Karlsruhe, Germany
{sla,aan,rst}@aifb.uni-karlsruhe.de

Stephan Grimm
FZI Research Center for Information Technologies, Karlsruhe, Germany
grimm@fzi.de

ABSTRACT

A key challenge for dynamic Web service selection is that Web services are typically highly configurable and service requesters often have dynamic preferences on service configurations. Current approaches, such as WS-Agreement, describe Web services by enumerating the various possible service configurations, an inefficient approach when dealing with numerous service attributes with large value spaces. We model Web service configurations and associated prices and preferences more compactly using utility function policies, which also allows us to draw from multi-attribute decision theory methods to develop an algorithm for optimal service selection. In this paper, we present an OWL ontology for the specification of configurable Web service offers and requests, and a flexible and extensible framework for optimal service selection that combines declarative logic-based matching rules with optimization methods, such as linear programming. Assuming additive price/preference functions, experimental results indicate that our algorithm introduces an overhead of only around 2 sec. compared to random service selection, while giving optimal results. The overhead, as percentage of total time, decreases as the number of offers and configurations increase.

Categories and Subject Descriptors: H.3.5 [Information Systems]: On-line Information Services – *Web-based services*; H.3.3 [Information Systems]: Information Storage and Retrieval – *Selection process*.

General Terms: Algorithms, Languages, Economics.

Keywords: Web Services, Customisation, Preference-based Service Selection.

1. INTRODUCTION

Web service discovery and selection have been extensively studied in recent years. As the set of available Web services may not be known a priori, may change frequently or service requester requirements and preferences may change, the problem of dynamic Web service selection is a fundamental one. Considerable research and industry effort has focussed on the (semantic) description of Web services, leading to standards such as WSDL [36], WSMO [9] and OWL-S [8]. One of the key open challenges is performing dynamic service selection for highly configurable Web services with dynamic user preferences. Web services are typically highly configurable, with significant service customisation possibilities and a choice of quality-of-service (QoS) properties, e.g. delivery/response times, naturally each with its own price. Customisation is critical for them to be able to differentiate themselves from competing Web services and offer a better service experience to their customers. Service requesters themselves have certain preferences on which service configurations they want to use and their preferences may change dynamically. Given their significance, there is a strong need to support customisable services. However, we lack efficient methods for the representation and matching of configurable services.

Current approaches to modelling Web service configurations and requester preferences, such as WS-Agreement [11], enumerate the various possible service configurations, which is inefficient when dealing with multiple service attributes and their values. For example, a service described by five attributes, each with five possible values, already leads to 3125 different configurations. Given this combinatorial increase, a functional description of service attributes and their associated prices or preferences would be more appropriate.

In this paper, we model service configurations and associated preferences compactly using utility function policies [17, 20], which provide a declarative mechanism for efficiently attaching price information to attribute values. This allows us to draw from the vast literature on efficient multi-attribute decision theory methods to develop an algorithm for optimal service selection. In addition, in order to be able to compare service attributes correctly, we need to describe them in a way that captures their semantics. We use ontologies to describe services attributes and their values semantically, and extend the resulting semantic representation to include offers and requests. This allows us to define appropriate attribute value matching rules for each kind of attribute.

The contributions of this work are three-fold. First, we develop an OWL ontology for configurable Web service offers and requests that can represent (execution-) context-dependent user preferences for functional and non-functional (e.g. QoS) properties within a standards-based specification language, thus extending current semantic Web service description frameworks, such as OWL-S and WSMO. Second, we present an optimal service selection mechanism in the context of this framework and demonstrate its feasibility by analysing its performance. For large numbers of highly configurable offers, given the assumption of additive functions, experimental results indicate that the algorithm introduces an overhead of only 2 sec. compared to random selection, while giving optimal results. This represents a 35% slowdown at 1000 offers and 1600 configurations per offer, which only decreases as the number of configurations rises. Third, our framework enables flexible matching by specifying declarative logic-based matching rules rather than hardcoding the matching algorithm as is usual. By providing a declarative mechanism that integrates optimization techniques, such as linear programming, we achieve computational tractability while obtaining a flexible system where different optimization and matching algorithms can be seamlessly plugged in.

In the following, we first discuss the requirements for a preference-

WWW 2007, May 8–12, 2007, Banff, Alberta, Canada.
ACM 978-1-59593-654-7/07/0005.

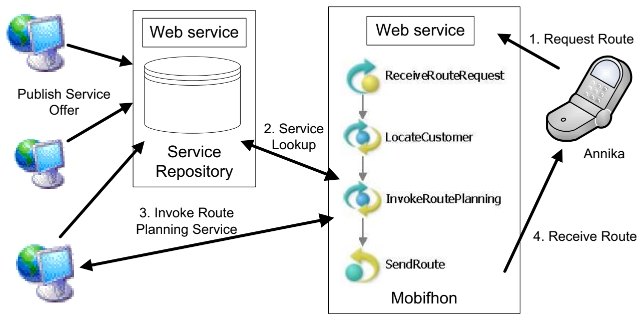


Figure 1: Example for dynamic Web service binding.

based service selection framework informally through a scenario in Section 2 and discuss the state-of-the-art with respect to these requirements. We then develop an abstract model that addresses the requirements for configurable services, preferences over configurations and service selection in Section 3. To enable the exchange of offers and requests in open heterogeneous environments, the abstract model is implemented in Section 4 using standard Web languages. Based on this formalization, matching and optimization rules for service selection are presented and evaluated (Section 5). In Section 6, we present a proof-of-concept implementation of our optimal service selection framework within the scenario developed in Section 2. We conclude the paper with thoughts on the feasibility of optimal service selection within current Web scenarios and point to future work directions.

2. REQUIREMENTS ANALYSIS

Consider Annika, a mobile phone user, who is currently in the city of Karlsruhe in Germany and wants to know the driving directions to Munich as soon as possible. Annika's mobile network operator, Mobifhon, provides route planning services for several countries to its customers, dynamically outsourced from third party route planning services on the Web, as sketched in Figure 1. Thus, the service selection takes place at Mobifhon's end. The service selection is therefore not constrained by the limited resources and partial connectivity of Annika's mobile phone, while allowing Mobifhon to aggregate demands and thus procure better discounts for services than if each customer were to transact individually. Mobifhon only sends the final route to Annika's mobile phone. For the sake of illustration, in the we have chosen a relatively general scenario for identifying the requirements, but one could imagine several simplifications, e.g. where the service repository is located with Mobifhon.

(R1) Service Configurations: The route planning services provide various kinds of routes (the fastest, the shortest etc.) with or without highways, identifying different kinds of attractions on the way. They also provide services at different levels of service quality. A more complex route planning, for example, will cost more than a simple route and similarly a quick response will cost more than a slower one. The various route planning services need to be able to describe their capabilities and configurations to Mobifhon, such that it can choose the appropriate one at the desired QoS level. Thus, our system must support *the description of various service configurations.*

Since WSDL lacks any support for modelling QoS characteristics of Web services, there have been several proposals to extend WSDL with concrete QoS metrics, e.g. the Web service Level Agreement (WSLA) project [14] and Web Service Modeling Language (WSML) [31]. By generalising beyond QoS attributes, XML query languages, like XQuery [39], and policy languages, like WS-Policy [38], enable the expression of constraints on arbitrary attributes. However, these approaches lack appropriate support for attaching prices and also preferences which are addressed by requirement (R2).

(R2) Context-dependent Preferences: Many of Mobifhon's customers have different preferences about the services they use, based on their current context, location, activities, etc. For example, Annika typically prefers to travel on highways but she is currently on vacation and wants to travel through scenic country roads, possibly making several stops at attractions on the way. Mobifhon chooses route planning services, taking into account the requirements and preferences of the user as well as its own preferences, e.g. on service characteristics such as availability, response time, supported encryption methods etc. Annika's preferences may also depend on her implicit context. For example, if she has an upcoming appointment in Munich the next day, she is more likely to prefer a short route than a long scenic route. In fact, her context-dependent preferences may be predefined, allowing Mobifhon to choose her preferences based on her context dynamically. To enable this, we need a way to *describe requests and preferences for particular service configurations declaratively in terms of their attributes.*

Requirement (R1) and (R2) are addressed by languages such as WS-Agreement [11] and the Web Service Offering Language (WSOL) [34], which introduce *classes of services* that roughly correspond to what we call service configurations and attach price and preference information to configurations. Both languages require the enumeration of the set of possible configurations. This is clearly inefficient for Web services whose attribute space is very large or even infinite (contradiction to (R4)). For example, the charges for route plans levied by the route planning service may decrease linearly with the desired response time. To cover such requirements, we use *Utility Function Policies* [17] to describe preferences and prices as a function of service attribute values, and we use declarative rules to model the context-sensitive nature of the preferences. This is similar to the approach presented in [3], where personalized service selection is realized by expressing preferences declaratively using SQL. However, since there is no rule support, context-dependent preferences cannot be expressed. In addition, matching algorithms required by (R3) are not supported there.

(R3) Semantic Web Service Descriptions: One of the advantages of using semantic description languages like OWL-S and WSMO is that one can use *logical reasoning*, in particular class subsumption, to bridge different levels of abstraction that occur when specifying requests and offers. Thus, if Annika has specified that she wants to know about attractions on the route, Mobifhon can identify route planning services with information about historical sites as being relevant. Annika may also not want to define preferences for all attributes. She may not know which attributes are used by the services she is interested in and even if she did, it would be too tedious. For example, she may want to say that she generally prefers historical sites to museums without specifying which particular types of each she prefers and by how much. The service matching algorithm needs to match her general preference to the actual attractions information provided by individual services. By modelling attributes as classes in a Semantic Web language, we can classify them into *attribute hierarchies*. It would not be possible to rely on semantic reasoning alone for service discovery and selection, as others [18] have also argued, since this only results in a coarse ranking.

There have been previous efforts to augment OWL-S and WSMO with QoS extensions [15, 33]. In addition, [21, 35] propose dynamic binding for Web service compositions using semantic service descriptions. However, as with the Web service description languages, they cannot be used to describe complex functional relations. Recent research [12, 26] has tried to express such relations declaratively, without however investigating how the performance of service selection is affected by the modelling of preferences and matching rules. Thus, these approaches often fail with respect to (R4). We use a decidable fragment of the rule language SWRL [13] to express complex functional relations declaratively, and discuss the effects of modelling on the performance of selection.

(R4) Communication and Computational Efficiency: A lesser requirement, yet nonetheless critical for resource-constrained environments such as mobile services, is that the chosen representation be designed for *communication efficiency and computational tractability*. I.e., the request has to be expressed in an efficient way (e.g. by avoiding enumeration of all possible configurations) and the selection algorithm has to be efficient enough to enable runtime selection.

Policy languages are state-of-the-art for expressing Web service configurations. However, as already discussed, while alleviating the problem, they cannot solve it due to the exponential size of the attribute space. We circumvent the problem using functional representations (as suggested in [10, 4]) by introducing Utility Function Policies. There is a vast amount of work in economics, and particularly in operations research, addressing the computational efficiency of decision making algorithms. In the context of service selection, this is investigated in [5], focusing on the complexity of service selection with one time costs and e.g. in [40, 35] for service compositions. Like these approaches, our work utilises efficient optimization techniques for service selection, but also augments them with the required service description and matching models.

Based on these requirements, in the next section we develop an abstract model for representing and selecting configurable services.

3. ABSTRACT SELECTION MODEL

Web service selection is the problem of selecting the best offer made by a service provider given a request. In order to perform Web service selection, one requires (i) means for communicating service offers as well as requests to the other party and (ii) an algorithm for ranking the offers with respect to the request. Bidding languages are a well-established means for communicating requests and offers within economic literature.

Our abstract model essentially describes a Web service, Web service offers and requests and the Web service selection problem. We take a fairly abstract view of a *Web service* in our model and consider it to be fully described by *properties* $A_1 \ldots, A_l, \ldots, A_n$. Such properties might comprise service input and output, behavioural aspects of a service, QoS attributes, etc., thus covering existing Web service description approaches as well as fulfilling (R1). Such a general description of a Web service allows us to abstract from various existing Web service description frameworks, while simultaneously allowing us to utilise existing decision-theoretic algorithms for multi-attribute products.

During the execution of a Web service, each attribute is assigned a value. A set C of Web service *configurations* comprises all possible combinations of attribute values, i.e. $C = A_1 \times \cdots \times A_n$. For example, considering the attributes *Attractions*, *Highways* and *Response Time* a concrete configuration would be a service providing routes including highways and information about nearby attractions within 10 seconds. A specific value of A_l is denoted by a_{le}.

A Web service *contract* t_{ij} is defined as a tuple (c, π), where agent j provides a Web service with configuration c to a customer i at a price of $\pi \in \mathbb{R}$. Furthermore, let T_j denote the set of all contracts involving provider j, and T_i the set of contracts involving customer i. Not all possible contracts are acceptable to an agent, and thus, only subsets $T'_j \subseteq T_j$ and $T'_i \subseteq T_i$ are requested or offered, respectively.

3.1 Bidding Language

For our bidding language for highly configurable Web services (R1), we draw from bidding languages for multi-attribute products [10, 4]. In this context, a common technique to efficiently encode pricing information (R4) is the use of functions that represent the relationship between Web service configurations and their prices or utilities (as discussed in (R2)). This avoids the combinatorial explosion that results from adding price markups to each configuration.

Definition 1 (Web Service Offer). *An* offer *by a provider j is defined as a pair (C_j, P_j) of a set $C_j \subseteq C$ of configurations and a function $P_j : C \to \mathbb{R}$ mapping each configuration $c \in C_j$ to a real number that represents the price π of invoking service configuration c. As suggested by [4], the pricing function $P_j(c)$ is described by a base price p_j^{base} and an additive function that aggregates pricing functions for individual attributes:*

$$P_j(c) = p_j^{base} + \sum_{l=1}^{n} w_{jl} p_{jl}(a_l) \text{ with } \sum_{l=1}^{n} w_{jl} = 1 \qquad (1)$$

where p_{jl} represents the pricing function of provider j for a particular attribute A_l. The weights w_{jl} are used to adjust the influence of the different attributes on the price.

Thus an offer assigns an additive pricing function to a Web service description, mapping the configurations of the offer to a certain price. Analogously, we introduce a functional form for representing Web service requests. One major difference though is that a requester's willingness to pay might depend on a runtime specific context (R2). Therefore, we introduce a set $K = \delta_1 \times \cdots \times \delta_n$ of *execution contexts*, where the δ_i represent different context dimensions, such as current location of a mobile device, time of service execution, history of past transactions. Any $k \in K$ denotes a concrete execution context.

Definition 2 (Web Service Request). *A Web service request by requester i is defined as a pair (C_i, F_i) of a set $C_i \subseteq C$ of acceptable configurations and a function $F_i : C_i \times K \to \mathbb{R}$ that maps each configuration to a real number score depending on the execution context k. Due to* payment monotonicity *[10], i.e. $\forall \pi > \pi' : (c, \pi) \in T'_i \Rightarrow (c, \pi') \in T'_i$, we interpret $F_i(c, k)$ as the maximal price for which a customer is willing to carry out the trade, i.e. $T'_i = \{(c, \pi) \in T_i | \pi \leq F_i(c, k)\}$. F_i is an additive scoring function composed of the attribute-specific functions f_{il} and their relative weights w_{il}:*

$$F_i(c, k) = \begin{cases} \sum_{l=1}^{n} w_{il} f_{il}(a_{le}, k) & \text{if } c \in C_i, \\ -\infty & \text{otherwise.} \end{cases} \text{ with } \sum_{l=1}^{n} w_{il} = 1 \qquad (2)$$

A configuration which is not requested is scored as minus infinity.

Due to the additive form of the scoring function F_i, we have to assume mutual preferential independency [16] between the attributes in the scoring function. This holds if the utility of an attribute A_l does not depend on the value of another attribute. For example, the score for a certain guaranteed response time will not change if the type of indicated attractions changes.

3.2 Selection Mechanism

Service selection in the case of configurable services involves finding the best provider and her best offer. Therefore, we have to solve two maximization problems: first, the best contract for a given provider has to be identified, the so-called *Multiattribute Matching Problem* (MMP) [10]. Second, based on these results, the best provider can be chosen. Service selection often requires trade-offs between the various attributes of configurable services. For example, it can be hard to decide between a slow route planning service that provides a lot of detailed information about en route attractions and a fast service that provides only imprecise route information. In economic literature, multi-attribute utility theory (e.g. [16]) uses utility functions to make such decisions, mapping each alternative to a measure that can be used to rank the alternatives. In our case, the utility of a service configuration is given by a *quasi-linear* function representing the difference between the requester's preference score for the configuration and its price. The MMP is thus defined as follows:

Definition 3 (MMP). *Given a request* (C_i, F_i)*, an offer* (C_j, P_j) *and an execution context k, we solve the MMP by maximizing the requester's utility per service configuration. The solution of the MMP for a given requester i and provider j is referred to as* u_{ij} *in the following.*

$$u_{ij} = \max_{c \in C_i \cap C_j} F_i(c, k) - P_j(c) \tag{3}$$

Our assumptions of additive pricing and scoring functions allow us to simplify the MMP. In particular, we can decompose the calculation into individual subproblems which can be solved independently. The following equations (4-6) use this property to solve Equation 3 efficiently by reducing the number of iterations from $O(|\prod_l A_l|)$ to $O(\sum_l |A_l|)$. The binary decision variable x_{le} is associated with each attribute value and denotes whether the value is part of the best configuration. Equation 5 ensures that exactly one attribute value is selected for each attribute. Since the following integer programming formulation has a totally unimodular constraint matrix and only integers on the constraints' right-hand sides, the problem can be solved efficiently using the simplex algorithm [28].

$$\max \quad \sum_{l=1}^{n} \sum_{e=1}^{|A_l|} (w_{il} f_i(a_{le}, k) - w_{jl} p_j(a_{le})) x_{le} - p_j^{base} \tag{4}$$

$$s.t. \quad \sum_{e=1}^{|A_l|} x_{le} = 1 \quad \text{for } 0 < l \leq n, \tag{5}$$

$$x_{le} \in \{0, 1\} \quad \text{for } 0 < l \leq n, \quad 0 < e \leq |A_l| \tag{6}$$

In a second step, we have to find the best provider from the set of all offers. Obviously this implies that the best contract for each provider is known, i.e. the *MMP* is solved for each pair of a request and an offer. We can then determine the best provider by solving *Local Allocation Problem* (LAP).

Definition 4 (LAP). *Given a single request* (C_i, F_i) *and m offers* (C_j, P_j)*, the Local Allocation Problem can be solved by iterating over all offers and determining the maximal solution for MMP.*

$$\max_{j=1,\ldots,m} u_{ij} \tag{7}$$

Solving LAP is linear with respect to the number of offers and requires $O(m)$ steps. However, there are several scenarios where LAP is not sufficient for service selection. First, if we relax the requirement for quasi-linearity of the utility function, e.g. by allowing one time costs, the problem will get considerably more complex. It can be shown by reduction of the *Uncapacitated Facility Location Problem* that computing the optimal service in such scenarios is in $\mathbf{FP^{NP}}$ [5]. Second, for the problem formulation in this section, we assume that offered services are always available for all requesters and that possible resource limitations are handled at the provider side, e.g. by adapting the guaranteed service levels or by increasing server capacity. LAP also needs to be extended to handle scarce resources. This is done, e.g. in [32] using a double auction or in [7] by means of scheduling algorithms. Third, LAP could be generalized for entire service compositions such as in [40]. The techniques presented later in this paper can also be applied to these new problems, requiring only the rewrite of a single rule to change the selection strategy.

4. ONTOLOGY-BASED REPRESENTATION

Given the abstract selection model above, we now focus on implementing this model using existing standards and tools for the open and heterogenous Web environment. We use the Web Ontology Language OWL [37] together with its rule extension SWRL [13] to implement our service selection framework, which allows us to perform sophisticated matchmaking and ranking of services by means of logical inferencing. We build on well-known notions of matching for Semantic Web services, such as subsumption-based "plugin" or "exact" matches [27] and develop a flexible and extensible framework of declarative matching and optimization rules.

4.1 Ontology Formalism

OWL is an ontology language standardized by the World Wide Web Consortium (W3C) [37] and is based on the description logic (DL) formalism [2]. Due to its close connection to DL it facilitates logical inferencing and allows to derive conclusions from an ontology that have not been stated explicitly. We briefly review some of the modelling constructs of OWL using its abstract syntax.

The main elements of OWL are *individuals*, *properties* that relate individuals to each other and *classes* that group together individuals which share some common characteristics. Classes as well as properties can be put into subsumption hierarchies. Furthermore, OWL allows for describing classes in terms of complex *class constructors* that pose restrictions on the properties of a class. For example, the statement `Class(`*BigCity* `partial restriction(`*connectedTo* `someValuesFrom` *Highway*`))` describes the class of big cities, which are connected to some Highway. The keyword `partial` means that any big city is connected to some highway, but not any city connected to a highway is also necessarily big, which would be achieved by using the keyword `complete` instead. Subclass relationship can be expressed by a statement like `SubClassOf(`*BigCity InterestingCity*`)`, saying that any big city is also interesting. Individuals can be related to classes and assigned values by a statement like `Individual(`*Munich* `type(`*BigCity*`) value(`*locatedIn Germany*`) value(`*population 1314551*`))`. Besides introducing Munich as a big German city, this statement also includes a data value for the city's population, which is supported by OWL for various datatypes such as integer or string.

An OWL ontology consists of statements like the ones above, considered logical axioms from which an agent can draw logical consequences. For example, given an ontology O consisting of the above statements, it follows that Munich is an interesting city, which is denoted by $O \models InterestingCity((Munich))$.

For the declarative formulation of matching directives in form of rules, we require additional modelling primitives not provided by OWL. We use the Semantic Web Rule Language (SWRL) [13] which allows us to combine rule approaches with OWL. We restrict ourselves to a fragment of SWRL called *DL-safe* rules[1] [24], which is more relevant for practical applications due to its tractability and support by inference engines such as KAON2[2]. For the notation of rules we rely on a standard first-order implication syntax.

4.2 Specification of Offers and Requests

In this section we present an ontology-based modelling approach for Web service offers and requests which is in line with Definition 1 and 2 of our abstract model. For the reader's convenience we present the most parts of our ontological model informally via UML class diagrams, where UML classes correspond to OWL concepts, UML associations to object properties, UML inheritance to subconcept relations, UML attributes to OWL datatype properties and UML dependencies to OWL class instantiation [6].

[1]DL-safety restricts the application of rules to individuals that are explicitly mentioned in the ontology. However, this restriction does not affect the suitability of DL-safe rules in our selection scenario.
[2]`http://kaon2.semanticweb.org/`

Figure 2 shows a top-level view of our ontological model, which can be split into two conceptual levels: the upper part a) captures the elements of the abstract model introduced in Section 3, while the lower part b) exemplarily captures certain available domain ontologies that are plugged in for the matchmaking of attribute values.

Web Service Description Ontology. Recalling definitions 1 and 2, offers and requests specify a set C of supported configurations in our abstract model, which they map to prices and scores, respectively. This is captured in the ontological model shown in Figure 2 a) by the classes in the two boxes for description of Web services and policies. The classes *Offer* and *Request* are introduced as subclasses of the more general *Bid*, by which they are connected to the policies used to define their pricing and scoring functions. They represent the sets C_j and C_i of configurations for a provider or requester. Instead of relating offers and requests to configurations directly, as done in the abstract model, we introduce the intermediary concept *Service* to capture technical service-specific aspects. Offers and requests are then related to a service which in turn supports various configurations. Referring to pairs of attributes A_l and their values a_{le}, the class *Configuration* represents the combinations of attribute values that a provider or customer support. This upper part of our ontology can be seen as extensions of existing Web service description ontologies such as OWL-S or WSMO by using these ontologies to define the type of *Attributes*. For instance, by introducing the concepts *Input*, *Output*, *Result*, etc. as specialisation of *Attribute* our ontology can be aligned to the OWL-S profile.

To give an example, recall our mobile phone scenario, where our user Annika requests route planning from Karlsruhe to Munich in at most 30 seconds while she wants nearby castles to be indicated along the route. For convenience, we illustrate the instantiation of elements for service description, such as *Configuration*, by an intuitive notation of pairs of attributes and their values, while we use OWL abstract syntax for details concerning the attribute values in domain ontologies. Colon-separated namespace prefixes indicate the origin of an entity in a domain ontology. An example of a supported configuration c_i for a service that Mobifhon launches as a request based on the above parameters could look as follows.

$$c_i = (\begin{array}{l} ServiceType = scl:RoutePlanningService \\ StartPoint = geo:Karlsruhe \\ EndPoint = geo:Munich \\ Attractions = tourism:Castle \\ ResponseTime = 30\ sec \quad) \end{array}$$

On the other hand, an example for a configuration c_j supported by an appropriate provider could look like this.

$$c_j = (\begin{array}{l} ServiceType = ServiceSupportingNavigation \\ StartPoint = geo:Germany \\ EndPoint = geo:Germany \\ Attractions = tourism:CulturalAttraction \\ ResponseTime = 1\ min \quad) \end{array}$$

```
equivalentClasses(ServiceSupportingNavigation intersectionOf(
scl:Service restriction(scl:supports someValuesFrom(scl:Navigation))))
```

Definition 1 and 2 introduce a compact functional form for expressing pricing and scoring information. This avoids introducing a separate class *Price* to model the tertiary relation between *Price*, *Configuration* and *Offer*/*Request* explicitly. Although such an approach would be most natural, it would result in a significant modelling overhead and does hardly scale up, as shown in our previous work [19]. For modelling functions we introduce the notion of *Policies*. Generally policies are declarative rules that guide the decision making of an agent. We use a specific class of policies, called utility-function policies [17], which allow for representing trade-offs between different Web service attributes by mapping their values to a comparable quantitative measure. Approaches on how such policies are expressed via an ontology are discussed in [20]. Namely there are three modelling techniques: *Point Based Functions*, *Piecewise Linear Functions* and *Pattern-based Functions*. To illustrate the idea, the concept *Point Based Function* is introduced in more detail.

A *Point Based Function* can be used for discrete attributes and is modeled by specifying sets of *Points* that explicitly map attribute values a_{le} referred to in an *Attribute Value Pair* to a *price* $f_{il}(a_{le})$ or $p_{jl}(a_{le})$. To indicate the *Attribute* for which a certain *Policy* is applicable, the *isAssignedTo*-relation is introduced that points to one of the *Attributes* in the Web service description. Coming back to our example, assume Annika generally prefers cultural attractions to sports events with the only exception that she hates museums. We can model such a preference structure by instantiating a point-based policy function assigned to the attribute *Attractions*. Mapping a utility of 1, 0.5 and 0 to the three alternatives results in the following specification of the function's component f_{il} for this particular attribute.

a_{le}	$f_{il}(a_{le})$
CultureWithoutMuseum	1
tourism : *SportsEvent*	0.5
tourism : *Museum*	0

```
equivalentClasses(CultureWithoutMuseum intersectionOf(
  tourism:CulturalAttraction complementOf(tourism.Museum)))
```

The table maps alternative values for the attribute *Attractions* to the utility values that specify Annika's preferences.

Domain Ontologies for Attribute Values. The attributes $A_1, \ldots, A_n$ in the abstract model represent generic characteristics of a service and can potentially originate from a given domain, depending on the kinds of services to be described. In our framework, we support this by plugging in various domain ontologies – depicted in Figure 2 b) – that describe attribute values, such as a classification of service types or geographic or tourism knowledge for attributes like *StartPoint* or *Attractions*. During the matchmaking process, this knowledge is, for example, used to detect that a service supporting navigation is equivalent to a route planning service, that Munich is in Germany, or that a castle is a historic site.

Assuming appropriate domain ontologies are available and agreed by providers and customers, they are linked to our ontology by their elements, such as the class *InformationService* or the individual *Munich*, being instances of the class *AttributeValue*. The value for the attribute *EndPoint* would be a URI like `http://geo.owl#Munich` that points to a location in a geographic ontology[3].

In our example, the notions of "route planning service" and "service that supports navigation" are captured in the service classification ontology O_{scl} that states them to be equivalent.

```
Ontology( O_scl
  Class(RoutePlanningService complete intersectionOf(
    Service restriction(supports someValuesFrom(Navigation))))
  ...)
```

Also the values "Karlsruhe", "Munich" and "Germany" are covered in a domain ontology, namely in the geographical ontology O_{geo}, where the two cities are stated to be located in Germany.

```
Ontology( O_geo
  Individual(Karlsruhe type(City) value(locatedIn Germany))
  Individual(Munich type(City) value(locatedIn Germany))
  ...)
```

Moreover, the values "cultural attraction" and "castle" for the attribute *Attractions* are related by subsumption in the ontology $O_{tourism}$ that describes notions of leisure.

```
Ontology( O_tourism
  subClassOf(HistoricSite CulturalAttraction)
  subClassOf(Castle HistoricSite) ...)
```

[3]Notice, that the OWL-Full language variant supports metamodelling, i.e. an URI can denote a class and an individual at the same time. Although metamodelling is outside the DL formalism, KAON2 can handle such URIs to a certain extend [22].

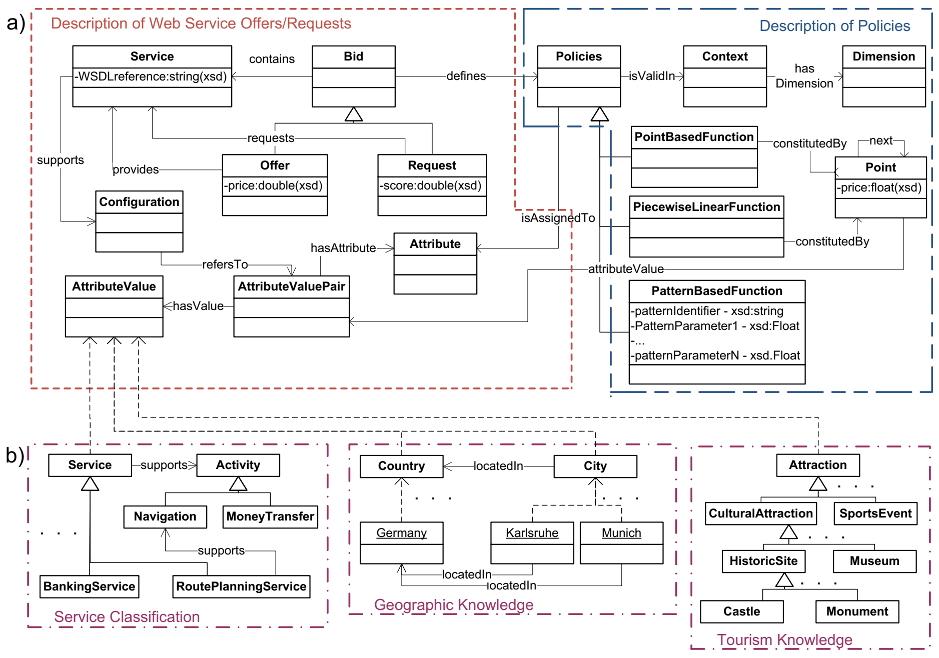


Figure 2: Ontology for representing Web service offers and requests

5. SERVICE SELECTION

Having shown how descriptions of configurable Web services offers and requests are captured by an ontological model, we now describe how the actual selection of a service is carried out by solving LAP using logical inferencing on the ontological elements introduced above.

In order to derive a ranked list of the offer instances from the knowledge base, we formulate a query that refers to a *Request* instance r containing preferences and to an instance k representing the current *Context*. In addition, to reduce the number of offers that have to be ranked we can add mandatory conditions directly to the query using the SPARQL FILTER element. This is exemplified in Query 8, where only services that provide a guaranteed response time of less than 20 sec. are retrieved.

```
PREFIX ex: < http://example.org/ns# >
SELECT ?O , ?U WHERE {
?O  ex:provides  ?S . ?S  ex:supports  ?C . ?C  ex:refersTo  ?A .
?A  ex:hasAttribute ex:ResponseTime ;  ex:hasValue  ?V .
FILTER (?V < 20) .  EVALUATE mmp(r, ?O, c, ?U) .}
ORDER BY DESC(?U)                                                   (8)
```

Conceptually answering such a query can be broken down into two highly connected parts: (i) determining matches between offers and requests by comparing the respective service attributes, and (ii) ranking the various configurations provided in the offer according to the preferences in the request. In the following, we first discuss how matching of attribute values is realized and show how these matching rules are applied to define variants of optimization predicate *mmp* used in Query 8. Subsequently, we evaluate these algorithms in terms of performance and discuss their applicability for service selection.

5.1 Matching

The comparison between a requested attribute value and an offered attribute value is fundamental. In the context of matchmaking in the Semantic Web, various techniques have been proposed for comparing the characteristics of two semantically annotated services. The most widely used approach for descriptions based on OWL classes is to use DL inferencing, distinguishing between several notions of match based on subsumption or concept satisfiability. For two OWL classes C_R and C_O that represent attribute values of a request or an offer, the degrees of match proposed in [27] are: *exact* if C_R and C_O are equivalent, *plugin* if C_R is a subclass of C_O, *subsumes* if C_O is a subclass of C_R, *intersect* if the conjunction of C_R and C_O is satisfiable, and *fail* if the conjunction of C_R and C_O is unsatisfiable.

We support these notions of match in our framework, and also allow for others by including customisable matching predicates into the service selection algorithm. In fact, since we use a declarative formalism to describe how attribute values are matched, a user can bring in arbitrarily complex matchmaking behaviour expressed in rules which facilitates the adaption of the selection component to changing service descriptions (e.g. with new attributes). Contrarily, other approaches use hard-coded algorithms to process the results of attribute matching that are specific to e.g. input/output matching, as done in [27]. Since we keep attributes in Web service configurations rather generic, the way in which two attribute values are compared strongly depends on the domain of interest they originate from and on the way in which they are represented in there.

In our example, the values of the attributes *ServiceType* and *IndicatedAttractions* are concepts in an ontology, and thus, the formerly described degrees of match apply to them and can be used for their comparison. The following rule definition specifies the matching predicate for service types, requiring them to yield an exact match.

$$\begin{aligned} &match(?P_1, ?P_2) \leftarrow hasAttribute(?P_1, ServiceType),\\ &\quad hasValue(?P_1, ?V_1), hasAttribute(?P_2, ServiceType),\\ &\quad hasValue(?P_2, ?V_2), exact(?V_1, ?V_2) \end{aligned} \tag{9}$$

In our example, the scl : *RoutePlanningService* indeed yields an exact match with the provided *ServiceSupportingNavigation*, since O_{scl} entails their equivalence. The attribute *ServiceType* itself is an instance of the class *Attribute* in the model shown in Figure 2. Analogously, values of the attribute *Attractions* could be matched using the predicate $plugin(x, y)$ instead of $exact(x, y)$, and again the provided attribute value would match the requested one, since $O_{tourism} \models$ `subClassOf(`*Castle CulturalAttraction*`)`.

The values of the attributes *StartPoint* and *EndPoint* from the example represent individual locations in a geographic ontology and require a different treatment. Here the modeller specifies the customised matching behavior for location attributes by introducing a class *Location* as a subclass of *Attribute*, of which *StartPoint* and *EndPoint* are instances. The appropriate matching behaviour is then captured by the following rule.

$$\begin{aligned} match(?P_1, ?P_2) \leftarrow\ & hasAttribute(?P_1, ?A_1), hasValue(?P_1, ?V_1), \\ & hasAttribute(?P_2, ?A_2), hasValue(?P_2, ?V_2), Location(?A_1), \\ & Location(?A_2), matchLocation(?V_1, ?V_2) \end{aligned} \tag{10}$$

$$matchLocation(x, y) = \begin{cases} true & O_{geo} \models locatedIn(x, y) \\ false & otherwise \end{cases} \tag{11}$$

The predicate *matchLocation* is realized as a builtin using a separate call to a description logic reasoner, just as in $exact(x, y)$ or $plugin(x, y)$ before. In the example, the offered value would again match the request, since Karlsruhe and Munich are both located in Germany according to O_{geo}.

Finally, there are attributes which do require a complex matching in terms of logical reasoning, but where a simple and efficient string comparison or arithmetic calculations are sufficient. In this case, the modeler of a matching rule can include predefined builtin predicates defined in SWRL. From our example, the QoS attribute *ResponseTime* falls into this category, and is processed according to the following rule specification.

$$\begin{aligned} match(?P_1, ?P_2) \leftarrow\ & hasAttribute(?P_1, ?A_1), hasValue(?P_1, ?V_1), \\ & hasAttribute(?P_2, ?A_2), hasValue(?P_2, ?V_2), QoSAttribut(?A_1), \\ & QoSAttribut(?A_2), equals(?V_1, ?V_2) \end{aligned} \tag{12}$$

Also here a subclass of *Attribute*, namely *QoSAttribute*, is introduced to enable the specification of the matching behavior for all QoS attributes, such as *ResponseTime*, by a single rule.

Based on definitions above, we can define a shortcut for matching two arbitrary configurations as follows:

$$compare(?C_1, ?C_2) \leftarrow \bigwedge_{l=1,\dots,n} attrCompare(A_l, ?C_1, ?C_2) \tag{13}$$

$$\begin{aligned} attrCompare(?A, ?C_1, ?C_2) \leftarrow\ & refersTo(?C_1, ?P_1), refersTo(?C_2, ?P_2), \\ & hasAttribute(?P_1, ?A), hasAttribute(?P_2, ?A), match(?P_1, ?P_2) \end{aligned} \tag{14}$$

5.2 Ranking

In the following, we show how the matching predicates introduced in the previous section can be used within the optimization rule *mmp* in order to determine the utility measure for the individual attribute values. We define three alternative variants of the *mmp*-predicate, which considerably differ in their underlying assumptions, applicability and performance characteristics. While the fist variant [V1] implements the ranking based on enumerating the configurations (Equation 3), [V2] solves MMP on per attribute basis using Equation 4-6. [V3] goes a step beyond [V2] by utilizing the linear program formulation.

[V1] This variant implements Equation 3, where a ranking of all offers and configurations is derived by evaluating all possible configurations for each offer according to a request. We can model the problem purely based on DL-safe rules using some standard SWRL builtin functions. Rule 15 calculates the difference between score and price of each *Configuration* that is supported by an *Offer* as well as asked for in the *Request*. The *compare*-predicate defined in Rule 14 is used to match two configurations.

$$\begin{aligned} mmp(?R, ?O, ?K, ?U) \leftarrow\ & provides(?O, ?S1), supports(?S1, ?C1), \\ & requests(?R, ?S2), provides(?S2, ?C2), compare(?C1, ?C2), \\ & price(?O, ?C1, ?P), score(?R, ?C2, ?K, ?S), sub(?S, ?P, ?U) \end{aligned} \tag{15}$$

Since pricing and scoring information are not explicitly given, the two predicates *price* and *score* are used to calculate this information based on the *Policies* defined in the offer or request, respectively. Rule 16 calculates the *score* $F_i(c, k)$ by evaluating $f_{il}(a_{le})$ for all a_{le} provided in a configuration c. Attribute values are matched by means of the matching predicate defined in the previous section. The *price*-relation is defined analogously, but without the context-dependency represented by the relation *isValidIn*.

Algorithm 1 Determine optimal attribute value for two policies

```
function optFkt(Policy $f_1$, Policy $f_2$)
    SELECT ?U WHERE {
        $f_1$ constitutedBy ?P1 . $f_2$ constitutedBy ?P2 .
        ?P1 attributeValue ?V1 ; price ?X .
        ?P2 attributeValue ?V2 ; price ?Y .
        ?V1 match ?V2 . EVALUATE ?U := dif(?X,?Y) .
    } ORDER BY DESC(?U)
    return first element of result set
```

$$\begin{aligned} score(?R, ?C, ?K, ?U) \leftarrow \bigwedge_{l=1,\dots,n} & (refersTo(?C, ?AVP_l), \\ & hasAttribute(?AVP_l, ?A_l), defines(?R, ?F_l), isAssignedTo(?F_l, A_l), \\ & isValidIn(?F_l, ?K), constitutedBy(?F_l, ?P_l), price(?P_l, V_l), \\ & attributeValue(?P_l, ?AV_l), match(?AVP_l, ?AV_l)), \\ & sum(?V_1, \dots, ?V_n, ?U) \end{aligned} \tag{16}$$

Advantages of this approach are that one can get a full ranking of all configurations, which might be required in some applications. Furthermore, it can be modelled purely based on standard modelling primitives provided by OWL-DL and SWRL. However, the disadvantages are also evident. Since the approach is based on enumerating all configurations, a finite number of configurations is required and thus the approach is not suitable in the presence of continuous attributes. As already discussed in Section 3, another fundamental problem is the complexity with respect to the number of required utility calculations.

[V2] The second variant of the *mmp*-predicate implements the decomposed optimization algorithm described in Equation (4-6). In this context we utilize the additive structure of the pricing as well as scoring functions: the optimal value for each attribute is determined separately and the overall price/score is calculated based on these measures. Equation 17 determines the utility of an offer according to request in a specific execution context.

$$\begin{aligned} mmp(?R, ?O, ?K, ?U) \leftarrow\ & Request(?R), Offer(?O) \\ & \bigwedge_{l=1,\dots,n} (defines(?R, ?F_l), isValidIn(?F_l, ?K), isAssignedTo(?F_l, ?A_l), \\ & defines(?O, ?P_l), isAssignedTo(?P_l, ?A_l), optFkt(?F_l, ?P_l, ?U_l)), \\ & sum(?U_1, \dots, ?U_n, ?U) \end{aligned} \tag{17}$$

Since the calculation of the optimal value for a certain attribute requires iterating over an unknown number of attribute values (instances), the calculation cannot be directly expressed in SWRL. We thus use a builtin function, called *optFkt*, to determine the attribute value a_{le} maximizing the utility $f_{il} - g_{jl}$ of attribute l. Algorithm 1 shows the implementation of the builtin-predicate specifically for Point-based Functions. In the predicate *optFkt* for each attribute the requester and provider policies are queried from the knowledge base and the attribute value leading to the maximal utility is determined. A major advantage of the approach is that this query uses the *match*-predicate defined in the ontology. Thus, the correct matching algorithm is used for each attribute automatically and the implementation of the builtin is domain independent.

[V3] The third variant of the algorithm implements also the decomposed ranking algorithm described in Equation (4-6), but with some additional optimizations.

$$\begin{aligned} mmp(?R, ?O?, ?C, ?U) \quad \leftarrow \quad & Request(?R), Offer(?O), \\ & optLP(?R, ?O, ?C, ?U) \end{aligned} \tag{18}$$

This time we use a linear programm to calculate the optimal attribute value. The calculation is encapsulated within the builtin *optLP* (Algorithm 2). The builtin performs a query to get the relevant utilities for the attribute values. This is done again by utilising

Algorithm 2 Optimization built-in using Linear Programming

```
function optLP(Request r, Offer o, Context k, Utility u)
    resultList := SELECT ?A, ?V1, ?U WHERE {
      o defines ?F1 . r defines ?F2 .
      ?F1 constitutedBy ?P1 ; isAssignedTo ?A .
      ?F2 constitutedBy ?P2 ; isValidIn k; isAssignedTo ?A .
      ?P1 attributeValue ?V1 ; price ?X .
      ?P2 attributeValue ?V2 ; price ?Y .
      ?V1 match ?V2 . EVALUATE ?U := dif(?X,?Y) .}
    Initialize $U = (u_{le})_{l=1...n,e=1...\max_l |A_l|}$ with $(A_l, a_{le}, u_{le}) \in resultList$
    determine $u = \max\{\langle U, X\rangle | \forall l \in \{1, \dots, n\} : \sum_{e=1}^{|A_l|} x_{le} = 1, x \in \{0,1\}\}$
    return u for given Offer o and Request r
```

the *match*-predicate from the ontology. The optimization problem is constructed and solved using a standard optimization library.[4] This approach has the advantage that we can use the efficient implementations for solving integer linear programs provided by standard tools.

In contrast to the first optimization algorithm, variant [V2] and [V3] can be easily adapted to handle continuous attributes by introducing appropriate builtins for *optFkt* and *optLP*. However, it is not possible to get a ranked list enumerating all offers and configurations, as it is possible using the first approach. Nevertheless, for most applications determining the ranked list of offers is sufficient.

In the next section we compare the different modelling approaches with respect to their performance in the selection process.

5.3 Performance Evaluation

Having introduced an approach for preference-based selection of configurable Web services, the question arises how this increased expressivity influences the performance of the selection process. In particular, we are interested in the trade-off between performance and optimality. Therefore, the three selection variants introduced in Section 5.2 are compared to an algorithm that randomly selects an offer and configuration. All algorithms are evaluated for varying number of offers in the knowledge base and varying numbers of configurations per offer. Each of these settings is evaluated by means of a simulation. Only settings with string matching rules have been used. Performance evaluations of query answering with more complex matching rules is a complementary question and has already been elaborated in [23] for KAON2. For each setting, instances of offers, requests and contexts are randomly generated using a uniform distribution and stored in the knowledge base. Then SPARQL-queries are generated according to Equation 8 (without any FILTER condition) referring to a specific execution context and request in the knowledge base. 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. For each setting the average query time is determined based on 20 simulation runs. Using this simulation setup the following issues are addressed:

How does the performance change when moving towards preference- and context-aware selection strategies? How expensive is optimality? To investigate the additional time required for evaluating the offers and configurations according to preferences, we compare the most general optimal variant [V1] with a baseline algorithm that randomly selects an offer and corresponding configuration from the knowledge base. Figure 3 shows the interrelation between the number of offers in the knowledge base, the number of configurations in an offer and the resulting query time. In the first setting (Figure 3(a)) each bid contains exactly 100 configurations. Service selection can be done in less than a second for 2000 offers using the random algorithm, while [V1] requires about 17 seconds. As depicted in Figures 3(b) and 3(c), the gap increases further for more demanding settings. However, the random approach leads to a considerable loss in utility for the requester. Assuming a uniform distribution of the prices and scoring values in [0,1] and a reasonable number of offers (> 50) an optimal algorithm leads to a utility of almost 1 while the random algorithm results only in a average utility of 0.

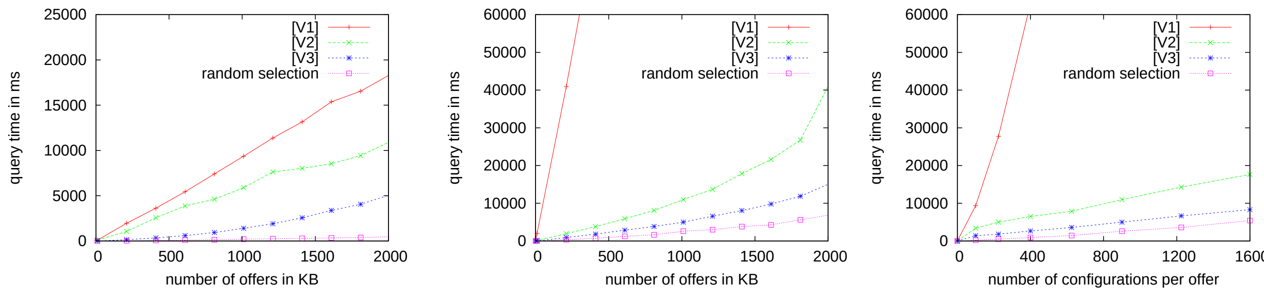
Can we improve the performance of optimal selection by constraining the bidding language? How does the performance of the optimal selection variants differ? Variants [V2] and [V3] of the *mmp*-predicate assume an additive structure of the pricing and scoring function. As discussed above, this allows a more efficient implementation. [V2] reduces the runtime compared to [V1] from 17 to 11 seconds in the first setting and from 477 to 41 in the second setting (both with 2000 offers). [V3] further reduces the runtime to 5 and 13 seconds, respectively. If we now compare this improved performance to the random algorithm, the cost of optimality is rather moderate. In particular, considering setting 3 with 1000 offers and 1600 configurations per offer, there is only a slowdown by 35% when moving from random selection to [V3] (Figure 3(c)). Comparing this number to smaller settings we can observe much greater slowdowns which, at first glance, seems contradictory. However, this observation can be explained by the fact that for large-scale scenarios (more that 1000 configurations per offer) query answering becomes the predominant factor compared to the optimization in [V3]. Since query answering is required for both algorithms, variant [V3] and random selection converge.

5.4 Discussion

In this section, we presented an flexible approach for assigning syntactic as well as semantic matching predicates to attributes. These predicates are automatically applied for matching attribute values in the optimization process. In general, the results of the performance evaluation are promising since the fastest approach allows ranking up to 2000 offers with a reasonable number of configurations below 15 seconds. Considering the fact that these 2000 offers all fulfill the mandatory conditions defined in the FILTER condition of Query 8, this can already be seen as a very large scenario. Moreover, we analysed the worst-case scenario where all offers provide all possible configurations and all attributes are discrete. Optimization on discrete attributes is more time consuming compared to the continuous case because techniques like differentiation are not applicable. Therefore, we expect better performance in a real-world use case. In our mobile scenario, only up to 20 different route planning providers might be available, whereas the number of possible configurations may easily exceed 1000. However, it is unlikely that all of them are offered by all providers.

As a further result of our performance study, it becomes clear that providing expressive means for modelling preferences as done in [12, 26] is not sufficient without ensuring that the way they are used allows for the implementation of efficient selection algorithms. Comparing the results for [V2] and [V3], we can identify the absence of the additivity assumption as the major source of complexity (improvement from [V1] to [V2]). Using an efficient implementation for solving the optimization problem provides a relatively minor improvement (improvement from [V2] to [V3]) in performance. Therefore, in many cases, especially if service selection has to be done at runtime, restricting the expressiveness of the bidding language is a viable way to considerably increase performance. Even if preferential independency does not hold exactly, additive functions often provide a good approximation [30]. If this simplification is not possible, other methods for improving the performance of [V1] can be introduced, e.g. a caching mechanism for prices and scores that reduces the number of rule evaluations [19].

[4]For our implementation we currently use the LP solver *lp_solve 5.5* (`http://lpsolve.sourceforge.net/5.5/`).



(a) 100 configurations per offer (b) 1000 configurations per offer (c) 1000 offers in KB

Figure 3: Performance of service selection.

A further conclusion is that in some very demanding settings, reducing the set of relevant offers is crucial. This can be realized by adding additional mandatory conditions through FILTER expressions.

6. PROTOTYPE

As a proof of concept, we implemented the algorithms presented in this paper in a framework consisting of two components.[5] The first is a server component that provides a repository for service offers and requests that can be queried via a Web service interface and the DL reasoner KAON2. KAON2 is chosen because it supports the logical fragment required for our offer and request descriptions, while being optimized for large-scale query answering [23]. This component corresponds to the service repository in Figure 1.

The second component is a client tool that facilitates the specification of Web service offers and requests by providing a GUI for specifying SPARQL-queries and utility function policies. Generally, offers are transferred to the server, whereas for requests the user decides whether they should also be stored as policies on the server to enable further reuse (cf. (R4)) or formulated directly as a query. The client is supplemented by a WS-BPEL engine [25] that allows the specification of service compositions. For example, the second component could be used by the network operator in our initial example to implement its application.

In order to implement dynamic binding of services, we utilise the distinction between ports and port types in WS-BPEL. This feature allows us to dynamically re-assign end points as long as the service candidates have an identical interface, i.e. port type. To support this, the client tool allows extending the process as follows: before each dynamic service invocation, a WS-BPEL *invoke*-operation for the selection service is introduced that provides the binding information for the following service. We realize the *Binding by Constraint* paradigm [29] by specifying a SPARQL query (such as Query 8) that is passed to the selection service. In this case, SPARQL provides a standardized language for identifying suitable services without referring to a concrete name or identifier. Parts of the query are generated at development time of the process, while others can be added dynamically at runtime. To illustrate this approach, Listing 1 shows an excerpt form the WS-BPEL process of the route planning scenario introduced in Section 2. In lines 2-4 the user's request is received, the contained *clientId* is passed to the location service, where the current country of the user is determined (lines 8-10). Then the SPARQL query, which statically refers to request *ns1:RequestOperatorX* containing the providers preferences, is extended by the context *location* (lines 12-15) and passed to the selection service which is invoked in lines 18-20. After receiving the address of the best service, the corresponding port is assigned

```
...                                                                          1
<receive name="receiveRoute" partnerLink="customer"                          2
    portType="client:Process" operation="initiate"                           3
    variable="routeRequest" createInstance="yes"/>                           4
<assign name="Assign_Query">                                                 5
    <copy> <from variable="routeRequest" part="user"/>                       6
        <to variable="clientID"/>                                            7
<invoke name="LocationCheck" partnerLink="LocationServicePLT"                8
    portType="ns1:LocationService" operation="executeQuery"                  9
    inputVariable="clientID" outputVariable="location"/>                     10
<assign name="Assign_Query">                                                 11
    <copy> <from expression=’"concat(string(\"SELECT ?O , ?U WHERE {         12
            EVALUATE mmp(ns1:RequestOperatorX, ?O,\" ),                      13
            bpws:getVariableData(’location’,’Country’),string(\",?U ) . }\"))"’/>  14
        <to variable="requestQuery" part="queryMessage"/>                    15
    </copy>                                                                  16
</assign>                                                                    17
<invoke name="ServiceInvoke" partnerLink="SelectionServicePLT"               18
    portType="ns1:SelectionService" operation="executeQuery"                 19
    inputVariable="requestQuery" outputVariable="responseQuery"/>            20
<assign name="Assign_Port">                                                  21
    <copy> <from variable="responseQuery" part="queryResult"/>               22
        <to partnerLink="RoutePlanner"/>                                     23
    </copy>                                                                  24
</assign>                                                                    25
<invoke name="RoutePlanningInvoke" partnerLink="RoutePlanner"                26
    portType="ns1:RoutePlanningService" operation="requestRoute"             27
    inputVariable="routeRequest" outputVariable="responseRoute"/>            28
...                                                                          29
```

Listing 1: Flexible binding in WS-BPEL

to the port type of the following partner link (lines 21-25) and the route planning service is invoked by passing the original user request containing the start and end point (lines 26-28).

In case service candidates have different interfaces, dynamic selection requires complex interface mappings. [29] present an approach for dynamic binding of services using reflection. However, this cannot be used directly for WS-BPEL.

7. CONCLUSION

In this paper, we have provided a formal and standards-based representation of Web service configurations and user preferences over these configurations meeting the requirements (R1) and (R2) introduced in Section 2. Our approach avoids enumerating offered/requested configurations and thus significantly reduces storage requirements and increases communication efficiency (R4). In addition, we have presented a service selection algorithm that seamlessly integrates syntactic as well as semantic matching (R3) with efficient optimization techniques. In contrast to other work in this area, we do not restrict ourselves to logical and/or similarity-based matching approaches, but allow customisable matching predicates that can be declaratively assigned to service attributes in a very flexible way. In order to quantify the overhead introduced by the additional expressivity and the optimality requirement, we evalu-

[5]More information about the implementation can be found at `http://km.aifb.uni-karlsruhe.de/projects/kaonws/`.

ated the performance of the different ranking algorithms, showing that an algorithm that implements the linear programming formulation of the optimization problem introduces only a small overhead compared to random selection. This holds particularly for large-scale scenarios with a high number of configurations per offer. Another important finding is that the performance depends crucially on the way offers and request are modelled. According to our evaluation, additive preference and price functions are required to dynamically select services in large-scale scenarios in a computationally tractable manner (R4). Finally, as a proof of concept we applied our framework for dynamic Web service selection in WS-BPEL.

As future work, we plan to move from selecting single services to service selection for an entire process. This can be realized simply by adding rules and builtin predicates implementing the more complex optimization algorithms (e.g. [40]). In addition, we plan to integrate behavioral matching rules as presented in [1], which would allow defining preferences over temporal properties of a service. For example, Annika might prefer services which provide the route information before paying. In terms of implementation, we plan to address the problem of dynamic binding in WS-BPEL beyond simple port re-assignments.

Acknowledgments. This research was partially supported by the German Research Foundation (DFG) in scope of the Research Training Group "Information Management and Market Engineering", by the European Commission under FP6-027595 "NEON", FP6-507482 "KnowledgeWeb", and FP6-507483 "DIP", and by the German BMBF project "SmartWeb". The expressed content is the view of the authors but not necessarily the view of any of the mentioned projects as a whole.

8. REFERENCES

[1] S. Agarwal and R. Studer. Automatic matchmaking of web services. In *5th Int. Conf. on Web Service*, Chicago,USA, 2006.

[2] 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.

[3] W.-T. Balke and M. Wagner. Towards personalized selection of web services. In *12th Int. WWW Conf.*, Budapest, Hungary, 2003.

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

[5] P. A. Bonatti and P. Festa. On optimal service selection. In *Proc. of the 14th Int. WWW Conf.*, New York, USA, 2005.

[6] S. Brockmans, R. Volz, A. Eberhart, and P. Löffler. Visual modeling of OWL DL ontologies using UML. In *Proc. of the 3rd Int. Semantic Web Conf.*, Hiroshima, Japan, 2004.

[7] P. Bruckner and S. Knust. *Complex Scheduling*. Springer, 2006.

[8] DAML Services Coalition. DAML-S: Web service description for the semantic web. In *1st Int. Semantic Web Conf.*, Sardinia, Italy, 2002.

[9] R. Dumitru, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel:. Web service modeling ontology. *Applied Ontology*, 1(1):77 – 106, 2005.

[10] Y. Engel, M. P. Wellman, and K. M. Lochner. Bid expressiveness and clearing algorithms in multiattribute double auctions. In *Proc. of the 7th ACM Conf. on e-Commerce*, New York, USA, 2006.

[11] Global Grid Forum. Grid Resource Allocation Agreement Protocol. Web Services Specification. Available from `http://www.ogf.org/Public_Comment_Docs/Documents/Oct-2006/WS-AgreementSpecificationDraftFinal_sp_tn_jpver_v2.pdf`, October 2006.

[12] B. Grosof and T. Poon. SweetDeal: Representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In *12th Int. WWW Conf.*, Budapest, Hungary, 2003.

[13] 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, 2004. W3C Submission.

[14] IBM Corporation. WSLA language specification, version 1.0. `http://www.research.ibm.com/wsla`, 2003.

[15] L. Kagal, T. Finin, and A. Joshi. Declarative Policies for Describing Web Service Capabilities and Constraints. In *W3C Workshop on Constraints and Capabilities for Web Services*, CA, USA, 2004.

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

[17] J. O. Kephart and W. E. Walsh. An artificial intelligence perspective on autonomic computing policies. In *5th IEEE Int. Workshop on Policies for Distributed Systems and Networks*, NY, USA, 2004.

[18] M. Klusch, B. Fries, M. Khalid, and K. Sycara. OWLS-MX: Hybrid Semantic Web Service Retrieval. In *1st Int. AAAI Fall Symposium on Agents and the Semantic Web*, Arlington, USA, 2005.

[19] S. Lamparter and A. Ankolekar. Automated selection of configurable web services. In *8. Int. Tagung Wirtschaftsinformatik*, Karlsruhe, Germany, 2007.

[20] 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 *8th Int. Conf. on Electronic Commerce*, New Brunswick, Canada, 2006.

[21] D. J. Mandell and S. McIlraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. In *2nd Int. Semantic Web Conf.*, FL, USA, 2003.

[22] B. Motik. On the properties of metamodeling in OWL. In *4th Int. Semantic Web Conf. (ISWC 2005)*, Galway, Ireland, 2005.

[23] B. Motik and U. Sattler. A comparison of reasoning techniques for querying large description logic aboxes. In *Proc. of the 13th Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning*, Phnom Penh, Cambodia, 2006.

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

[25] OASIS. Web Services Business Process Execution Language (WS-BPEL). `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel`, 2007. Version 2.0.

[26] N. Oldham, K. Verma, A. Sheth, and F. Hakimpour. Semantic WS-Agreement Partner Selection. In *15th Int. WWW Conf.*, Edinburgh, UK, 2006.

[27] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara. Semantic matching of web services capabilities. In *1st Int. Semantic Web Conference*, pages 333–347, Sardinia, Italy, 2002.

[28] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization*. Englewood Cliffs, N.J.: Prentice Hall, 1982.

[29] C. Pautasso and G. Alonso. Flexible binding for reusable composition of web services. In *Proc. of the 4th Workshop on Software Composition*, Edinburgh, Scotland, 2005.

[30] S. Russel and P. Norvig. *Artificial Intelligence - A Modern Approach*. Prentice Hall, 2nd edition, 2003.

[31] A. Sahai, V. Machiraju, M. Saya, A. v. Moorsel, and F. Casati. Automated SLA monitoring for web services. In *Proc. of 13th Int. Workshop on Distributed Systems*, Montreal, Canada, 2002.

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

[33] I. Toma, D. Foxvog, and M. C. Jaeger. Modeling QoS characteristics in WSMO. In *1st Workshop on Middleware for Service-oriented Computing*, New York, USA, 2006.

[34] V. Tosic, K. Patel, and B. Pagurek. WSOL - web service offerings language. In *CAiSE Workshop on Web Services, E-Business, and the Semantic Web*, Toronto, Canada, 2002.

[35] K. Verma, R. Akkiraju, R. Goodwin, P. Doshi, and J. Lee. On accommodating inter service dependencies in web process flow composition. In *AAAI Spring Symposium on SWS*, CA, USA, 2004.

[36] W3C. Web Services Definition Language (WSDL) 1.1. `http://www.w3.org/TR/wsdl`, 2001.

[37] W3C. Web Ontology Language (OWL). `http://www.w3.org/2004/OWL/`, 2004.

[38] W3C. Web Services Policy Framework 1.5. `http://www.w3.org/2002/ws/policy/`, July 2006.

[39] W3C. W3C XML Query (XQuery 1.0). `http://www.w3.org/XML/Query/`, January 2007.

[40] 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.