Semantic Specification and Evaluation of Bids in Web-based Markets

Steffen Lamparter [a,*] Anupriya Ankolekar [a] Daniel Oberle [c]
Rudi Studer [a] Christof Weinhardt [b]

[a] *Universität Karlsruhe (TH), Institute of Applied Informatics and Formal Description Methods (AIFB), Karlsruhe, Germany*

[b] *Universität Karlsruhe (TH), Institute of Information Systems and Management (IISM), Karlsruhe, Germany*

[c] *SAP Research, CEC Karlsruhe, Germany*

Accepted for publication in ELECTRONIC COMMERCE RESEARCH AND APPLICATIONS (ECRA).

Abstract

In recent years, electronic markets have gained much attention as institutions to allocate goods and services efficiently between buyers and sellers. By leveraging the Web as a global communication medium, electronic markets provide a platform that allows participants throughout the world to spontaneously exchange products in a flexible manner. However, ensuring interoperability and mutual understanding in such a highly dynamic and heterogenous environment can easily become very tricky, particularly if the services and goods involved are complex and described by multiple attributes. In this paper, we present a comprehensive ontology framework that allows the *specification* of bids for Web-based markets. By expressing utility function policies with a logic-based and standardized formalism, the framework enables a compact bid representation particularly for highly configurable goods and services while ensuring a high degree of interoperability. To facilitate matchmaking of offers and requests in the market, a method for *evaluating* bids based on logical reasoning is presented. In addition, as proof of concept, we show how the framework can be applied in a Web service selection scenario.

Key words: Electronic Markets, Ontology, Policy, Utility Theory, Bidding Language

* Corresponding author.
Email address: lamparter@aifb.uni-karlsruhe.de (Steffen Lamparter).

1 Introduction

Electronic markets are institutions that allow the exchange of goods and services between multiple participants through global communication networks, such as the Internet. In the process, Web-based markets create economic value for buyers, sellers, market intermediaries, and for the society at large [6]. For example, they reduce search costs, enable companies to automate their transactions with business partners all over the world, and facilitate product customization and aggregation. Generally, the design of market platforms involves two components [43]: (*i*) A bidding language which defines how offers and requests can be specified and submitted to the market mechanism. In this context, a *bid* is considered as "a statement of what one will give or take for something". [1] That means requests express the price someone is willing to pay and offers the price someone charges for a certain product. (*ii*) An outcome determination that evaluates the bids in order to calculate an allocation between offers and requests. Depending on the domain, this step might involve market mechanisms ranging from simple take-it-or-leave-it markets to mechanisms implementing complex negotiation and auction protocols.

When moving to a heterogenous Web environment where business partners dynamically join and leave the market and where participants might not know each other beforehand, designing a bidding language is a non-trivial task since it requires ensuring *mutual understanding* between different participants in the market. This problem is aggravated by the fact that many trading objects in the Web are *highly configurable*, described by different attributes and traded under different conditions. A prominent example for such highly configurable trading objects are Web-based services, which have become increasingly important with the emerging service-oriented computing paradigm in recent years. 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 and goods generally requires that both *seller pricing functions* as well as buyer *scoring (preference) functions* be taken into account.

The key contribution of this work is to develop a means for specification and evaluation of bids that enables compact representation of multi-attribute offers and requests while addressing the challenges that arise from a Web-based environment. For expressing pricing and preference functions we suggest combining a declarative policy approach with utility theory, which quantifies preferences by assigning cardinal valuations to each configuration. Such *utility function policies* enable

[1] Merriam-Webster Online

efficient specification of multi-attribute bids, while being exchangeable between different market participants due to their declarative nature.

In order to achieve interoperability and ensure mutual understanding, our technology of choice is ontologies. Given our assumption of heterogeneous market participants and global markets, it is quite likely that requests and offers will use differing descriptions for the same service or product. Going beyond XML-enabled interoperability, ontology-based descriptions of service or product attributes enable the comparison of requests and offers that are semantically related. Existing electronic markets do without ontology-based descriptions of products, but at the expense of rigidly requiring each participant to agree a priori to conform to the standard protocol and product descriptions of the market. Using ontologies within markets enables participants to describe their service or product the way they deem best, leaving it to the market to identify the best matches and thus facilitate interoperability. In addition, they make the market (participants) resilient to changes in the vocabularies or market mechanisms.

Before presenting our ontology framework, we discuss the requirements this framework has to meet and review related work to determine whether the requirements are already supported by existing approaches (Section 2). Subsequently in Section 3, we introduce the concept of utility function policies and the basic principles behind ontologies. Based on these foundations, an ontology framework for expressing configurable offers and requests is presented. The main building blocks of the framework are described in the following sections: the Core Policy Ontology that enables expressing utility function policies is introduced in Section 4 and the Core Ontology of Bids that extends the formalization for expressing offers and requests is presented in Section 5. Finally, as proof of concept, a concrete implementation of the framework in the Web service domain is presented in Section 6. Section 7 concludes the paper with a discussion and brief outlook.

2 Requirements Analysis and Related Work

In this section, we first introduce the main requirements for the specification of bids in a Web-based market (Section 2.1) and discuss whether existing approaches meet these requirements (Section 2.2).

2.1 Requirements

In this paper, we consider the problem of designing a mechanism for specification as well as evaluation of bids that enables the expression of highly configurable offers and requests in an efficient way, while being applicable in an open Web envi-

ronment. Such a mechanism has to meet several requirements, which are discussed in the following.

Multi-attribute Requests and Offers: First, since we are dealing with various configurable products, the bid specification has to support *multi-attribute requests and offers*, i.e. requests and offers that involve multiple attributes beyond just the price. Often these attributes considerably influence the price of a trading object in the market. For example, in the case of Web services quality of service attributes are in fact an essential part of the service description although they do not address the service functionality itself. The attributes that have to be described in this context can be discrete as well as continuous.

Expression of Functions: Second, pricing as well as scoring policies require the *expression of functions* that map configurations to a pricing or a preference structure, respectively. A functional form is essential particularly for highly configurable trading objects since enumeration of configurations is made difficult by the exponential size of the attribute space. For instance, a product described by 5 attributes, each with 5 attribute values, already involves over 3000 configurations. Moreover, preferences have to be measured on a cardinal scale, so that one can specify both an ordering between offers and an acceptance threshold for offers that satisfy the request to a certain degree.

Web-compliant Interoperability: Third, since offers have to be communicated to the buyer and/or requests to the seller (e.g. in a procurement auction), *interoperability* becomes an important issue. This is particularly crucial in open markets on the Web, where participants may use highly heterogenous data formats and participants as well as descriptions may change frequently. Therefore, a standardized syntax and semantics is essential that ensures valid matching in the market although requests and offers are often specified differently. For example, a service provider might specify that routes between all cities in Europe are supported, while a customer might look for a route between exactly two cities Karlsruhe and Munich. To bridge different levels of abstraction sophisticated logical inferencing mechanisms are required which, in this case, utilize the knowledge about cities being in countries and countries belonging to continents. Moreover, when implementing markets using Web infrastructure offers and request descriptions have to be *compliant to existing protocols and languages on the Web*, i.e. they should be serialized using XML/RDF, identify objects by Uniform Resource Identifiers (URI), feature modularization, etc.

2.2 Related Work

In this section, we present various existing approaches in electronic markets for modeling buyer preferences and seller offerings. Table 1 summarizes the approaches discussed in terms of which of the three requirements they support (indicated by

Table 1
Languages for specifying offers and requests.

Approach	**Requirement**		
	Multi-attribute Offers/Requests	Expressing Functions	Web-compliant Interoperability
EDI/EDIFACT	(✓)	(✓)	-
XML-based Languages	✓	-	(✓)
Ontology-based approaches	✓	-	✓
Rule-based Languages	✓	(✓)	(✓)
Product/Service Catalogs	-	-	✓
Bidding Languages for CA	✓	✓	-
CPML	✓	✓	(✓)
Our Approach	✓	✓	✓

check marks).

One of the first attempts to exchange order information within electronic markets were the Electronic Data Interchange (EDI) protocols (e.g. EDIFACT [60], X12 [2]), which serialize request and offer information according to a predefined format agreed upon by both communication parties. Thus, EDI could potentially be used to describe multi-attributive requests and offers with preference and pricing functions. However, these pairwise agreements were rarely based on any standards and turned out to be effort-intensive, highly domain-dependent and inflexible. Thus, EDI is not addressing the interoperability requirement.

More recent approaches, such as WS-Policy [66], EPAL [27] and XACML-based Web Service Policy Language (WSPL) [39], use XML (eXtensible Markup Language) [65] as a domain-independent syntax, to define constraints on attributes of configurable trading objects within the context of Web service agreements. However, they are not suitable for our purposes, because they only allow the expression of attribute value pairs and thus cannot be used to express seller pricing and buyer scoring functions. In addition, the meaning of XML annotations is defined in a natural language specification, which is not amenable to machine interpretation and supports ambiguous interpretation. Therefore, such approaches require extensive standardization efforts, which is also a major problem for ebXML [46]. WS-Agreement [3] is another XML-based specification that can be used to express different valuations for configurations. However, it supports only discrete attributes. An approach to extend WS-Agreement for expressing continuous functions is presented in [54]. The XML annotations still lack formal semantics and therefore do not provide the required interoperability.

One way to enable machine interpretation of buyer requests and seller offers is to specify them using a machine-interpretable ontology. Such an ontology consists of a set of vocabulary terms, with a well-defined semantics provided by logical axioms constraining the interpretation and well-formed use of the vocabulary terms. Based on this idea several proposals for semantically enabled matching in electronic markets have been presented [45,36,58,19]. The approaches improve semantic interoperability by utilizing logical descriptions contained in requests and offers. Depending on the logic used and on the concrete trading objects in the market, different notions of match can be defined. However, logical matching does typically lead only to a coarse preference structure over the different alternatives, e.g. [51] presents four different levels of match, namely *exact*, *plugin*, *subsumes* and *fail*. Therefore, [56,31] argue that pure logical matching is not sufficient and has to be augmented by "value reasoning". This is also the approach we follow in our work by introducing functional descriptions. Another problem with these approaches is that they do not directly support constraints on multiple attributes. Therefore, the ontology-based policy languages KAoS [61] and REI [28] can be used to extend the matching approaches in order to allow the definition of multi-attribute policies for representing constraints on attributes. However, these approaches are limited in that they always evaluate either to true or false thus cannot express the scoring or pricing functions required for configurable products.

More expressivity in this context is provided by rule languages such as SweetDeal [20] and DR-NEGOTIATE [57]. Both are rule-based approaches that use defeasible reasoning (i.e. Courteous Logic Programs or defeasible logic) to specify contracts or agent strategies, respectively. Similar to our approach they feature automatic reasoning based on a formal logic. However, although RuleML is available as a standard syntax, the semantics of the syntax is not yet standardized which hinders interoperability. In addition, while the underlying rule language might be capable of expressing utility-based policies, they do not provide the required policy specific modeling primitives directly, rather the rules for interpreting such policies have to be added manually by the user. In the DR-NEGOTIATE approach qualitative preferences are expressed via defeasible rules and priorities among them. While such an approach is suitable for ranking of alternatives, it is not possible to assess the absolute suitability of an alternative, which is important in case the best alternative is still not good enough (cf. Section 3.1). Most similar to our approach is the work presented in [50], where WS-Agreement is extended with an ontological model and preferences are expressed via a rule-language. However, similar to the previous approaches they use a non-standard rule language and do not elaborate on the structure of the preference rules.

A separate stream of work has focussed on developing highly expressive bidding languages for describing various kinds of attribute dependencies and valuations, particularly in the context of (combinatorial) auctions (e.g. [44], [11], [55]). However, they assume a closed environment and therefore, even if they do use XML-based bidding languages [9], they do not deal with interoperability issues. Many

B2B scenarios use standardized product and service taxonomies, such as UN/SPSC [59], CPV [15], or the MIT Process Handbook [37]. However, while these taxonomies are suitable for pure functional descriptions, their static hierarchical structure makes them inappropriate for multi-attribute descriptions since each new product configuration requires the introduction of a new subclass in the hierarchy. Therefore, they might be applicable to describe individual attributes, but not the entire multi-attribute trading object.

Recapitulating, two major streams of work can be identified. First, approaches coming from the area of market engineering that propose highly expressive bidding languages for complex electronic markets. However, they lack support for interoperability in a Web environment. Second, there are approaches addressing interoperability in heterogenous and dynamic environments such as the Web. These approaches typically do not focus on market-specific requirements. Our work unifies these two aspects in one coherent framework. We draw from utility theory to express scoring and pricing functions of market participants and thereby enable compact representation of request and offers. Furthermore, we show how these functions can be expressed declaratively with a standardized and Web-compliant ontology formalism.

3 Ontology-based Specification of Requests and Offers

In this section, we first introduce the principles of policy-based decision making focusing on the concept of utility function policies (Section 3.1). Such policies allow encoding preference and price information in an efficient and declarative way. In Section 3.2, we argue how interoperability in the Web can be realized by means of ontologies. Finally, relying on these concepts we present in Section 3.3 an ontology framework for compactly expressing configurable requests and offers.

3.1 Utility Function Policies

Policies are declarative rules that guide the decision making process by constraining the decision space, i.e. they specify which alternatives are allowed and which are not. The decision space can be divided into "indifference ranges" [14] that represent sets of alternatives with the same preference level. [53,30] refer to such policies as *goal policies*. However, when making a decision it is often not enough to know which alternatives are allowed, but rather which is the best alternative and how good the alternative is (e.g. the best alternative might still be not good enough). Therefore, we suggest combining a declarative policy approach with utility theory [29], which quantifies preferences by assigning cardinal valuations to each alternative. With such *utility function policies*, detailed distinctions in preferences can be

expressed, providing improved decision making between conflicting policies compared to traditional *goal policies* by explicitly specifying appropriate trade-offs between alternatives [30].

In the context of electronic markets, utility function policies can be used on buyer-side to specify preferences, assess the suitability of trading objects and derive a ranking of trading objects based on these preferences. Further, they allow the exchange of preferences with sellers which might be required, for instance, in procurement auctions or exchanges. Since the functional form avoids enumerating all configuration to attach prices, utility function policies enable the compact representation of pricing or cost functions on the seller-side and thus provide an efficient way of communicating pricing information to the customers. In the remainder of this paper, we denote rules that define the relation between configurations and prices defined by a seller as *pricing policies* and rules that define how much a buyer is willing to pay for a certain configuration as *scoring policies* (or buyer preferences).

For our policy specification we use the following utility model. Assume alternatives (e.g. configurable trading objects) are described by a set of attributes $A = \{A_1 \dots A_n\}$. Attribute values a_j of an attribute A_j are either discrete, $A_j \in \{a_{j1}, \dots, a_{jm}\}$, or continuous, $a_j \in [min_j, max_j]$. Then the cartesian product $C = A_1 \times \dots \times A_n$ defines the potential configuration space, where $c \in C$ refers to a particular configuration. Based on these definitions a preference structure is defined by the complete, transitive, and reflexive relation $\succeq$. For example, the configuration $c_1 \in C$ is preferred to $c_2 \in C$ if $c_1 \succeq c_2$. The preference structure can be derived from the value function $V : C \rightarrow \mathbb{R}$, where the following condition holds: $\forall c_a, c_b \in C : c_a \succeq c_b \Leftrightarrow V(c_a) \geq V(c_b)$.

In literature there is a broad stream of work about modeling value functions (e.g. [16,29,62,5]). The goal is to provide sufficient expressivity for modeling complex decisions, while keeping the elicitation and computation effort at an admissible level. In the most general case, a value function is directly defined as a function $V(a_1, \dots, a_n)$ mapping each configuration to a valuation. Since the number of configurations is exponential with respect to the number of attributes and their values, this approach is infeasible already for relatively small problems. Fortunately, preferences of a customer often have an underlying structure which is introduced by the independency of the attributes. Relying on this structure substantially improves compactness and analytic manipulability [62]. The most prominent approach in this context are additive models, where the valuation $V(c)$ is decomposed into several lower-dimensional functions. There are several well known approaches for doing this decomposition based on different structural assumptions. In the following, we shortly introduce the *additive* value model which has favorable computational properties but also requires very restrictive assumptions.

The *additive value function* is defined in Equation (1) below assuming *mutual pref-*

erential independency between the attributes [29]. The attributes $A_1, \ldots, A_n$ are considered mutually preferential independent if every subset of these attributes is *preferentially independent* of its complement. The set of attributes $X \subset A$ is preferentially independent from the set $Y \subset A$ with $X \cap Y = \emptyset$ only if for some y' the following condition holds: $\forall y, x', x'' : [(x', y') \geq (x'', y')] \Rightarrow [(x', y) \geq (x'', y)]$. Under this assumption, we can decompose the utility function $V(c)$ into the individual functions $v_j(a_j)$ of the attribute A_j. The overall value can be calculated by Equation (1), where λ_j represents the weighting factor of an attribute normalized in the range $[0, 1]$.

$$V(c) = \sum_{j=1}^{n} \lambda_j v_j(a_j), with \sum_{j=1}^{n} \lambda_j = 1 \tag{1}$$

However, in real markets the preferential independency often does not hold. In order to at least partly capture this, dependent attributes $A_k, \ldots, A_l \in A$ can be treated as one single attribute A_j^* in our model, where the utility function is modeled as a complex (higher dimensional) function $v_{j^*}(a_k, \ldots, a_l)$. Since this approach allows one attribute of the product A_j to influence several of the aggregated attributes A_j^*, we support the family of *generalized additive* value functions [16,5]. While the generalized model requires expressing high dimensional functions for the entire product, the additive model requires attaching an one dimensional function to each attribute of the product. Since in general determining value functions of an agent is rather difficult, we assume extensive methodology and tool support in the preference elicitation process (cf. [13]).

In the following, we show how utility function policies can be declaratively specified and evaluated in a Web environment.

3.2 Achieving Interoperability with Ontologies

In order to achieve interoperability we use *ontologies*. In recent years, ontologies became an important technology for knowledge sharing in distributed, heterogeneous environments, particularly in the context of the *Semantic Web* [63,7]. An ontology is a set of logical axioms that formally define a shared vocabulary [21]. By committing to a common ontology, software agents can make assertions or ask queries that are understood by the other agents. Moreover, such logic-based definitions come with executable calculi that enable data integration required in electronic markets for matching offers and requests (e.g. [36,58,45]). In the following, we briefly introduce the logical formalisms we use to specify bids, namely the *ontology language OWL-DL* and the *rule language SWRL*.

In order to guarantee mutual understanding, the underlying logic has to be standardized. The Web Ontology Language (OWL) standardized by the World Wide Web Consortium (W3C) [64] is the only standardized ontology language that is currently available. OWL ontologies can be serialized via RDF/XML syntax while resources are identified using URIs. Both features and the extensive modularization possibilities provided by OWL ensure compatibility with existing World Wide Web languages. OWL comes in three levels of expressiveness. We use the most expressive decidable fragment OWL-DL. It is based on a family of knowledge representation formalisms called *Description Logics (DL)* [4], where the meaning of the provided modeling constructs like concepts, relations, datatypes and individuals is formally defined via a model theoretic semantics, i.e. it is defined by relating the language syntax to a model consisting of a set of objects, denoted by a domain, and an interpretation function, which maps entities of the ontology to concrete entities in the domain [25]. Thereby, the meaning of an axiom defines certain constraints on the model. For example, we can define that the concept *GermanRoute* is a subconcept of *Route*, which captures routes starting from a German city and ending in a German city. Using the OWL abstract syntax [4] this can be written as follows: $\mathit{GermanRoute} \sqsubseteq \mathit{Route} \sqcap \exists \mathit{from}.\mathit{GermanCity} \sqcap \exists \mathit{to}.\mathit{GermanCity}$. From a set of such axioms conclusions can be derived that are not explicitly stated in the ontology, e.g. a subsumption hierarchy between concepts in the ontology can be constructed. This is particularly important for matchmaking in markets since offers and requests are usually described on different levels of abstraction, e.g. when looking for a route planning service for Germany also route planning service for entire Europe are relevant. In the remainder, DL axioms are denoted by $A1, \ldots, An$.

In order to define our ontology, we require additional modeling primitives not provided by OWL-DL, e.g. triangle relations between concepts. Such modeling constructs are provided by rule languages. The Semantic Web Rule Language (SWRL) [23,24] allows us to combine rule approaches with OWL. Since reasoning with knowledge bases that contain arbitrary SWRL expression usually becomes undecidable [23], we restrict ourself to the *DL-safe* fragment of SWRL [42], which is more relevant for practical applications due to its tractability and support by existing inference engines. In addition, SWRL provides an extensible set of built-in predicates that can be used for implementing operations such as arithmetic calculation, string comparisons or manipulations, etc. For the notation of rules we rely on the standard first-order implication syntax, where built-ins are identified by the prefix "swrlb". In the following, rules are labeled by $R1, \ldots, Rn$.

3.3 Ontology Framework for Web-based Markets

In this section, we present an ontology framework for modeling offers and requests containing utility function policies introduced in Section 3.1 with the formalism presented in Section 3.2. Our framework consists of several ontology modules.

Table 2
Upper level concepts from DOLCE, Descriptions and Situations (DnS), Ontology of Plans (OoP) and Ontology of Information Objects (OIO) that are used as modeling basis.

Module	**Concept label**	**Explanation**
DOLCE	Endurant	Static entities such as objects or substances
	Perdurant	Dynamic entities such as events or processes
	Quality	Basic entities that can be perceived or measured
	Region	Quality space
DnS	Description	Non-physical objects like plans, regulations, etc. defining Roles, Courses and Parameters
	Role	Descriptive entities that are played by Endurants (e.g. a customer that is played by a certain person)
	Course	Descriptive entities that sequence Perdurants (e.g. a service invocation which sequences concrete communication activities)
	Parameter	Descriptive entities that are valued by Regions like the age of customer
	Situation	Concrete real world state of affairs using ground entities from DOLCE
OoP	Task	Course that sequences Activities
	Activity	Perdurant that represents a complex action
OIO	InformationObject	Entities of abstract information like the content of a book or a story

These modules are arranged in three layers:

Top-level Ontology: As a modeling basis, we rely on the domain-independent upper-level foundational ontology. By capturing typical *ontology design patterns* (e.g. location in space and time), foundational ontologies provide basic concepts and associations for the structuring and formalization of application ontologies. Reusing these building blocks considerably reduces modeling effort. Furthermore, they provide precise concept definitions and a high axiomatization. Thereby, foundational ontologies facilitate the conceptual integration of different languages and thus ensure interoperability in heterogenous environments. As foundational ontology we use DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) [38]. DOLCE provides the ontology design patterns required for formalizing policies such as contextualization and is available in the ontology language we use. The DOLCE concepts that are directly used for align-

ment of our ontology are briefly introduced in Table 2. A detailed description of DOLCE and its modules is given in [38,17].

Core Ontology: As a second layer we add modules for describing offers and requests in electronic markets. These ontologies are specific for a certain purpose, but still domain independent. The first module is the *Core Policy Ontology* (CPO) formalizing the notion of utility function policies which is then used in the second module. This module is called *Core Ontology of Bids* and introduces general communication primitives for expressing the intentions of participants in the market.

Domain Ontology: While the first two layers contain domain-independent off-the-shelf ontologies, the third layer comprises ontologies for customizing the framework to specific domains (e.g. an ontology for modeling products and their attributes).

In this paper, we focus on the second level of our ontology framework. The Core Policy Ontology is described in Section 4 and the Core Bidding Ontology in Section 5. To illustrate the use of these ontologies we also introduce fragments of a domain ontology for route planning services as a running example.

4 Core Policy Ontology (CPO)

The Core Policy Ontology (CPO) provides primitives for specifying goal and utility function policies introduced in Section 3.1. The remainder of this section is structured as follows: First, we extend the DOLCE ground ontology by modeling primitives required for representing functions between attribute values and their individual valuation by a user (Section 4.1). Secondly, based on these functions, we show how the DOLCE ontology module Description & Situation is applied to model product configurations and policies (Section 4.2). In addition, we discuss in this section how configurations are evaluated according to the specified policies. Finally, Section 4.3 introduces a mechanism to specify and evaluate collections of policies.

Note that although in the following we focus on applying the policy ontology for specifying scoring and pricing functions in electronic markets, due to its generality it is not restricted to this domain. In fact, it can be used for a wide range of multi-attribute decision problems, e.g. to define preferences over agent strategies or penalties in electronic contracts.

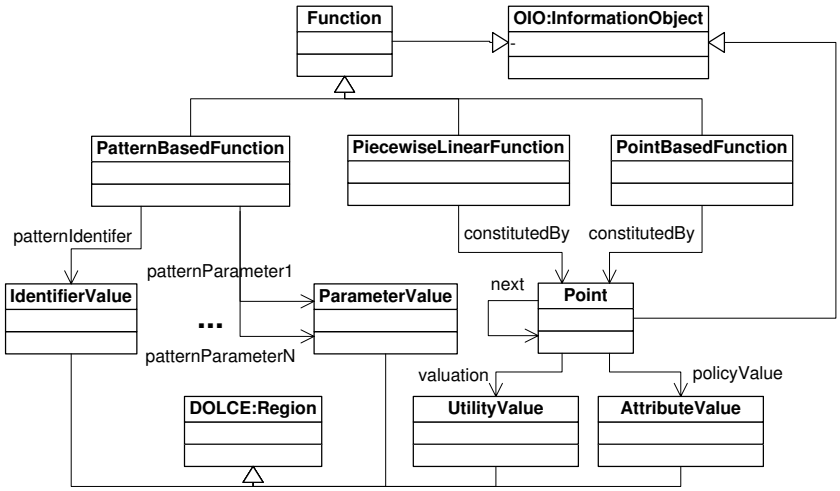

Fig. 1. Modeling value functions [2]

4.1 Valuation Functions

As discussed in Section 3.1, utility function policies are expressed via functions $V : C \rightarrow \mathbb{R}$ that map configurations $c \in C$ to a corresponding valuation between 0 (or $-\infty$) and 1, where a valuation of $-\infty$ refers to forbidden alternatives and a valuation of 1 to the optimal alternative [35]. We now show how the fundamental concepts formalized in DOLCE can be extended to allow expressing valuation functions.

As depicted in Figure 1, a *Function* [3] is a specialization of *OIO:InformationObject* which represents abstract information that exists in time and is realized by some entity [17]. Currently our framework supports three ways of defining functions: (*i*) *Functions* can be modeled by specifying sets of points that explicitly map attribute values to valuations. This is particularly relevant for nominal attributes. (*ii*) We allow to extend these points to piecewise linear value functions, which is important when dealing with continuous attribute values, such as the response time of a service. (*iii*) Thirdly, we allow reusing typical function patterns, which are mapped to predefined, parameterized valuation rules. Note that such patterns are not restricted to piecewise linear functions since all mathematical operators provided by the rule language can be used. The different ways of declarative modeling functions are discussed next in more detail.

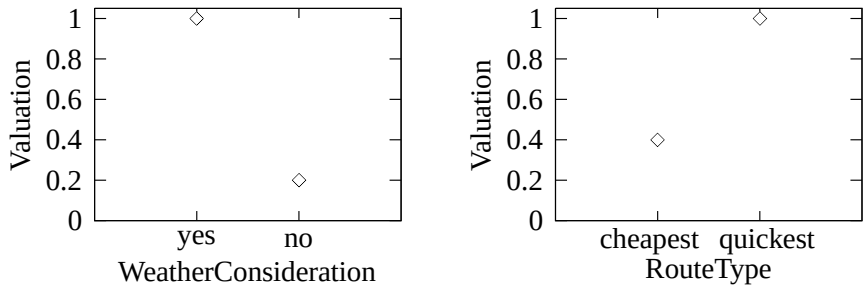

Fig. 2. Example of a point-based value function

4.1.1 Point-based Functions

As depicted in Figure 1, *PointBasedFunctions* are *Functions* that are *constitutedBy* a set of *Points*. Each *Point* has a property *policyValue* referring to an *AttributeValue* and a property *valuation* that assigns exactly one *UtilityValue* to this attribute value. [4] An *AttributeValue* is a specialization of *DOLCE:Region* that defines which attribute values a certain attribute may adopt, e.g. the attribute *WeatherConsideration* in the route planning example requires the *DOLCE:Region WeatherConsiderationValue* containing the elements {"*yes*", "*no*"}. Similarly, the *DOLCE:Region UtilityValue* comprises the range $[0, 1]$ and $-\infty$.

In our route planing example introduced in Section 1, a requester might specify her preferences with respect to the service property *WeatherConsideration* by a *PointBasedFunction*, which is *constitutedBy* two instances of *Point* with ("*yes*", 1) and ("*no*", 0.2). Thus, the requester would highly prefer weather information to be taken into account, but has some small use for routes calculated without weather information. Similarly, the preferences for the attribute *RouteType* calculation can be defined with *Points* ("*quickest*", 1) and cheapest ("*cheapest*", 0.4). These mappings are illustrated in Figure 2.

In order to evaluate this function, additional axioms are required that more closely define the semantics of the concepts *PointBasedFunction* and *Point* as well as their relations. Rule (R1) below defines how the *valuation* of a certain *policyValue* x can be determined based on the specification of the *PointBasedFunction* f. For this purpose we iterate over all *Points* constituting the function and compare their *policyValue* to the desired attribute value x.

---

[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 objects to OWL individuals [12].

[3] Concepts and relations of the ontology are written in *italics*. All concepts and relations imported from other ontologies are labeled with the corresponding namespace. Sometimes concept names in the text are used in plural to improve the readability.

[4] Note that in case we have dependent attributes and thus complex value functions $v_{j^*}(x_k, \ldots, x_l)$ (cf. Section 3.1) each *Point* might have several *policyValue* relations, i.e. $policyValue_k, \ldots, policyValue_l$.

$$degree(f, x, v) \leftarrow PointBasedFunction(f), constitutedBy(f, p), \\ policyValue(p, pv), match(x, pv), valuation(p, v) \quad \text{(R1)}$$

The comparison of attribute values is realized by the *match*-predicate. This predicate has to be customizable since the way attributes are compared depends on the domain of interest, i.e. on the concrete attribute. In order to keep Rule (R1) applicable for all attributes, we specify this in a separate matching rule. For example, considering the attribute *WeatherConsideration*, for matching the attribute values a simple string matching predicate as provided with the built-in *equals* is sufficient. Rule (R2) illustrates this by defining the matching rule for the attribute *WeatherConsideration*.

$$match(x, y) \leftarrow WeatherConsiderationValue(x), \\ WeatherConsiderationValue(y), swrlb{:}equals(x, y) \quad \text{(R2)}$$

Unfortunately, in many cases attribute values have to be described in a more complex way beyond simple strings or numbers, e.g. to express subclass relations between attribute values. In such cases it might be required to model attribute values as concepts in OWL. Since in our ontology they are modeled as individuals a meta-modeling approach is required where a URI can be treated as concept as well as instance. [5] This allows us to specify preferences on a more abstract level and thus avoids enumerating all possible attribute values.

For example, consider an attribute *IndicatedAttraction* that specifies which types of attractions along the route can be suggested by a certain service. In this case the corresponding value space *IndicatedAttractionValue* might comprise the alternatives *CulturalAttraction*, *HistoricSite*, *Museum* and *Castle* which are all related to each other. In particular, *CulturalAttraction* can be seen as a class containing all other values. *HistoricSite*, in turn, comprises *Castles* but not *Museums*. Consequently, a scoring function mapping historic sites to a valuation of 0.8 has to assign the same value to information about *Castles* along the route (although this might not be specified explicitly). Such a behavior can be realized by defining a *Point* that maps the *policyValue HistoricSite* to 0.8 and another *Point* that maps everything else to zero using the concept definition $Attraction \sqcap \neg HistoricSite$. Similar to the attributes above we can define a matching rule for the attribute *IndicatedAttraction* by replacing the built-in implementing string matching with a built-in that features DL subsumption checking between two concepts.

[5] Although such an approach is outside of the ontology formalism and part of OWL-Full, many reasoners such as KAON2 can handle meta-modeling to some extent [40].

$$match(x, y) \leftarrow IndicatedAttraction(x), IndicatedAttraction(y), \\ swrlb{:}subsumes(x, y) \tag{R3}$$

Several other matching variants have been proposed in literature (e.g. [36,45,8]). We support these different notions of match by providing a flexible framework that can be customized via declarative matching rules.

4.1.2 Piecewise Linear Functions

In order to support definition of *Functions* on continuous properties too, we introduce *PiecewiseLinearFunctions* as shown in Figure 1. Continuous attribute exhibit a natural ordering between the attribute values which can be utilized for specifying the function. We therefore extend the previous approach by the property *next* between two *Points* with adjacent attribute values.

Such adjacent *Points* can be connected by straight lines forming a piecewise linear value function as depicted in Figure 3.

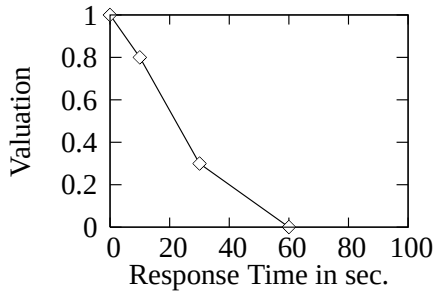

Fig. 3. Example of a piecewise linear value function

For every line between the *Points* (x_1, y_1) and (x_2, y_2) as well as a given *PolicyValue* x, we calculate the valuation υ as follows.

$$\upsilon = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1, & \text{if } x_1 \leq x < x_2 \\ 0, & \text{otherwise} \end{cases}$$

This equation is formalized by a predicate $cal(v, x, x_1, y_1, x_2, y_2)$. This predicate can be realized either directly by means of a built-in or by exploiting the math as well as the comparison built-in predicates provided by the rule language. [6]

[6] Although predicates with arity higher than two cannot be modeled with the formalism at hand directly, many reasoning tools support them. Moreover, techniques for reifying higher arity predicates are well known [26].

Using this predicate, Rule (R4) defines the valuation of a certain attribute value x (as Rule R1 does for *PointBasedFunctions*). The rule makes sure that only adjacent *Points* are considered in the calculation.

$$\begin{aligned} degree(f, x, v) \leftarrow & PiecewiseLinearFunction(f), \\ & \bigwedge_{i\in\{1,2\}} (constitutedBy(f, p_i), policyValue(p_i, pv_i), \\ & valuation(p_i, v_i)), next(p_1, p_2), \\ & cal(v, x, pv_1, v_1, pv_2, v_2) \end{aligned} \tag{R4}$$

As an example, let us assume the *Function* for the attribute *Response Time* of the route planing service is given by a *PiecewiseLinearFunction* with the *Points (0, 1), (10, .8), (30, .3), (60, 0)* as depicted in Figure 3. Now, we can easily find out which *valuation v* a certain *policyValue x* is assigned to. The predicate $cal_v(v, x, x_1, y_1, x_2, y_2)$ is true iff the *policyValue x* is between two adjacent *Points* (x_1, y_1) and (x_2, y_2) and the *valuation* equals *v*. For instance, for a *Response Time* of 20 sec. cal_v evaluates the straight line connecting the adjacent Points *(10, .8)* and *(30, .3),* which results in a *Valuation v* of .675.

4.1.3 Pattern-based Functions

Alternatively, value functions for continuous attributes can be modeled by means of *PatternBasedFunctions*. This type refers to functions like $u_{p_1,p_2}(x) = p_1 e^{p_2 x}$, where p_1 and p_2 represent parameters that can be used to adapt the function. In our ontology, these *Functions* are specified through parameterized predicates which are identified by *patternIdentifiers*. A *patternIdentifier* points to a *DOLCE:Region IdentifierValue* that uniquely refers to a specific rule predicate. This predicate is denoted *pattern*. A *patternParameter* defines how a specific parameter of the *pattern*-predicate has to be set. For allowing an arbitrary number of parameters in a rule, universal quantification over instances of *patternParameter* would be necessary in the body of the rule. Since this is not expressible with our rule language, the different parameters are modeled as separate properties in the ontology, viz. *patternParameter1,…, patternParameterN*. Of course, this restricts the modeling approach as the maximal number of parameters has to be fixed at ontology design time. However, we believe that keeping the logic decidable justifies this limitation.

As shown in the example below (Rule (R5)), each *pattern* is identified by a hard-coded internal string. This is required to specify, which pattern is assigned to a certain attribute in the ontology. Thus, in order to find out which *pattern*-predicate is applicable, the *patternIdentifer* specified in the policy is handed over to the predicate by using the first argument and then it is compared to the internal identifier. If

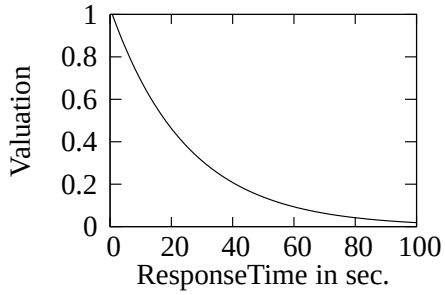

Fig. 4. Example of a pattern-based valuation function

the two strings are identical the predicate is applied to calculate the *valuation* of a certain *policyValue*.

As an example, we again focus on the attribute *ResponseTime* of the route planning service. In many scenarios the dependency between configurations and prices or valuations are given by functions. Assume the preferences for *Response Time* are given by the exponential function $u_{p_1,p_2}(x) = p_1 e^{p_2 x}$ with the *patternParameters* $p_1 = 1.03$ and $p_2 = -.04$ (Figure 4). Rule (R5) formalizes the pattern. The internal identifier in this example is *'id:exp'*. The corresponding comparison is done by the built-in *equals*, which is satisfied if the first argument is the same as the second argument.

$$\begin{aligned} &\mathit{pattern}(v, \mathit{id}, x, p_1, \ldots, p_n) \leftarrow \\ &\qquad \mathit{String}(\mathit{id}), \mathit{PolicyValue}(x), \mathit{Valuation}(v), \\ &\qquad \mathit{swrlb{:}equals}(\mathit{id}, \text{“}\mathit{id{:}exp}\text{”}), \mathit{swrlb{:}multiply}(t_1, p_2, x), \\ &\qquad \mathit{swrlb{:}pow}(t_2, \text{“2.70481”}, t_1), \mathit{swrlb{:}multiply}(v, p_1, t_2) \end{aligned} \tag{R5}$$

SWRL supports a wide range of mathematical built-in predicates (cf. [24]) and thus nearly all functions can be supported. As in our example, these functions are typically parameterized only by a rather small number of parameters. Therefore, we believe that there are few practical implications of defining the maximal number of parameters at ontology design time.

Based on the definition of the *pattern*-predicate we can define the *degree* of a certain attribute value according to a *PatternBasedFunction* using the following rule.

$$\begin{aligned} \mathit{degree}(f, x, v) \leftarrow &\mathit{PatternBasedFunction}(f), \mathit{patternIdentifier}(f, \mathit{id}), \\ &\mathit{patternParameter}(f, p_1), \ldots, \mathit{patternparameter}(f, p_n), \\ &\mathit{pattern}(v, \mathit{id}, x, p_1, \ldots, p_n) \end{aligned} \tag{R6}$$

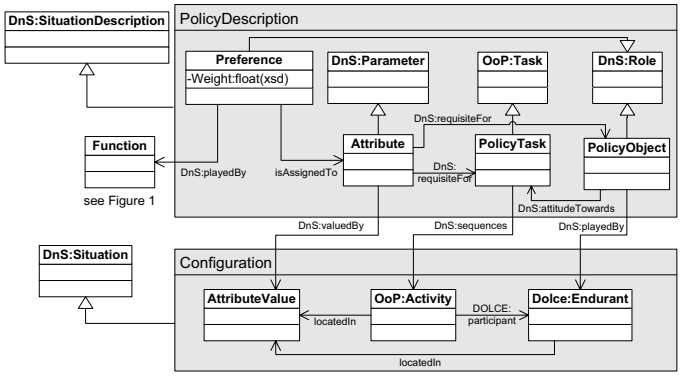

Fig. 5. Policy description framework. To improve the readability we illustrate certain relations by plotting UML classes within other UML classes: The class *PolicyDescription* has a *DnS:defines*-relation and the class *Configuration* a *DnS:settingFor*-relation to each contained class.

Based on the notion of *Functions* introduced above, we show in the following how they are used to define and reason about policies.

4.2 Modeling Policies and Configurations

As discussed in Section 3.1, we formalize preferences of a user as well as pricing information of a provider in a functional form by means of policies. For instance, a price-conscious user might prefer a cheap service although the service has a rather slow response time, whereas a time-conscious user might accept any costs for a fast service. Hence, policies can be seen as different views on a certain configuration. For modeling such views we use and specialize the DOLCE module Descriptions & Situations (DnS) which provides a basic theory of contextualization [17]. Hence, a certain configuration can be considered as more or less desirable depending on the scoring policies of a buyer or a configuration can be priced differently depending on the pricing policies of a seller.

When using DnS with DOLCE, we distinguish between DOLCE *ground entities* that form a *DnS:Situation* and *descriptive entities* composed in a *DnS:SituationDescription*, i.e. the view in which *Situations* are interpreted. As depicted in Figure 5, we specialize the *DnS:SituationDescription* to a *PolicyDescription* that can be used to evaluate concrete *Configurations* which are modeled as special kind of *Situations*. This distinction enables us, for example, to talk about products as roles on an abstract level, i.e. independent from the concrete entities that play the role. For instance, a certain product configuration can be evaluated in the light of either a pricing policy of the seller or the preferences of a user depending on the point of view.

In the following, we describe how such *Configurations* and *PolicyDescriptions* are modeled and then show how the evaluation of policies is carried out.

4.2.1 Configuration

In a first step, we define the ground entities that constitute a *DnS:Situation*. In our context, such *DnS:Situations* reflect multi-attribute real-world objects or activities. In a concrete *DnS:Situation* these products have one distinct configuration. Recall in Section 3.1 we defined the set of configurations C as the cartesian product of the attributes $C = A_1 \times \cdots \times A_n$. Hence, as shown in Figure 5, we model *Configuration* as a subclass of *DnS:Situation* that exactly defines one configuration $c \in C$ of a product. Since there are various different ways of describing products, a generic approach is used in this work, where concrete objects and activities are represented by instances of *DOLCE:Endurant* and *OoP:Activity*, respectively. Attributes of *DOLCE:Endurants* and *OoP:Activities* are modeled via the *locatedIn* property that points to a value range represented by the *DOLCE:Region AttributeValue* [18]. The following axioms capture this notion by ensuring that each *Configuration* comprises at least one multi-attribute object (Axiom (A1)). Axiom (A2) ensures that each *AttributeValue* belongs to exactly one *DOLCE:Endurant* or *OoP:Activity*.

$$\begin{aligned} \textit{Configuration} \sqsubseteq & \textit{DnS:Situation} \sqcap \exists \textit{DnS:defines}.(\textit{DOLCE:Endurant} \sqcup \\ & \textit{OoP:Activity}) \sqcap \exists \textit{DnS:defines.AttributeValue} \qquad \text{(A1)} \\ \textit{AttributeValue} \sqsubseteq & \textit{DnS:Parameter} \sqcap =_1 \textit{locatedIn}^{-}.(\textit{OoP:Activity} \sqcup \\ & \textit{DOLCE:Endurant}) \qquad \text{(A2)} \end{aligned}$$

Coming back to our example, a configurable Web service can be modeled by a combination of *DOLCE:Endurants* and *OoP:Activities* [48]. Hereby, specializations of *OoP:Activities* capture *ServiceActivities* like *RoutePlanningActivity*. Specializations of *DOLCE:Endurants* represent the objects involved in such a *ServiceActivity* (e.g. inputs and outputs). A *RoutePlanningActivity* might have several *DOLCE:Qualities* that are located in specializations of *AttributeValue* such as *WeatherConsiderationValue*, *IndicatedAttractionValue*, *ResponseTimeValue* or *AvailabilityValue*. In addition, the *RoutePlanningActivity* involves a *ServiceOutput* which specializes *DOLCE:Endurant* (or more specifically OIO:InformationObject)). *ServiceOutput* is associated to a *RouteTypeValue* that defines whether the output is the cheapest or the fastest route.

4.2.2 Policy Description

In a second step, we define views on the ground entities defined in Section 4.2.1. This is realized by specializing the descriptive entities *DnS:Roles*, *DnS:Courses*, *DnS:Parameters*, and *DnS:SituationDescriptions*. As depicted in Figure 5, policies are modeled as specialization of *DnS:SituationDescription*, called *PolicyDescriptions*, which have to *DnS:define* a *PolicyObject* or *PolicyTask* representing the entity

on which the policy is defined, e.g. this could be a certain type of good or a service. Since *PolicyObjects* and *PolicyTasks* are modeled as specialization of *DnS:Roles* and *OoP:Tasks*, policies can be defined on an abstract level without referring to a concrete *DOLCE:Endurant* or *Activity*. For instance, policies can be defined for a certain service category (specialization of *OoP:Task*) such as route planning services in general. Then all *OoP:Activities* that fulfill the task of route planning in a certain *Situation* are evaluated according to the policy. Axiom (A3) formally defines a *PolicyDescription*. It ensures that at least one entity is constrained by means of the *DnS:Parameter Attribute*. Moreover, each *Attribute* that is introduced has to constrain exactly one *PolicyObject* or *PolicyTask* which can be realized by means of the *DnS:requisiteFor*-relation (Axiom A4).

$$\begin{aligned} \textit{PolicyDescription} \sqsubseteq \textit{DnS:SitutationDescription} \sqcap \exists \textit{DnS:defines}.(\textit{PolicyObject} \\ \sqcup \textit{PolicyTask}) \sqcap \exists \textit{DnS:defines.Attribute} \qquad \text{(A3)} \\ \textit{Attribute} \sqsubseteq \textit{DnS:Parameter} \sqcap =_1 \textit{DnS:requisiteFor}.(\textit{PolicyObject} \\ \sqcup \textit{PolicyTask}) \qquad \text{(A4)} \end{aligned}$$

Up to now a *PolicyDescription* can be used to define constraints on certain properties of an entity. This is exactly what we consider as Goal policies. A similar approach is used in [49] for expressing policies such as access rights. However, as discussed in Section 3.1, utility function policies generalize this approach by addressing the fact that configurations are preferred to varying degrees depending on the concrete attribute values. Therefore, the *DnS:Role Preference* is introduced. *Preferences* can be assigned to an *Attribute* (via the *isAssignedTo*-relation) to enable modeling additive preference functions. Thus, preference structures on attributes are imposed by *Functions*. As discussed in Section 4.1, *Functions* are *OIO:InformationObjects*. They play the role of *Preferences* in a *PolicyDescription* and define how *policyValues* are mapped to *valuations*. That means, a policy defines which *Function* should be used in which context (i.e. for which attribute). Besides defining *Functions*, *Preferences* also define the relative importance of the given *Attribute* via the *DnS:Parameter Weight* and the DnS:Region *WeightValue* (omitted in Figure 5). Consequently, *Preferences* are formally defined as follows:

$$\begin{aligned} \textit{Preference} \sqsubseteq \textit{DnS:Role} \sqcap =_1 \textit{DnS:playedBy.Function} \sqcap \\ =_1 \textit{DnS:requisites.Weight} \qquad \text{(A5)} \end{aligned}$$

As an example, consider the policy that specifies the scoring function for the property response time of a Web service. To express this we instantiate *OoP:Task* by a *WebServiceTask*. In addition, an *Attribute ResponseTime* is introduced that represents a constraint (*DnS:requisiteFor*) that has to be fulfilled by *WebServiceTask*.

In order to define preferences over all possible attribute values, *ResponseTime* is *DnS:valuedBy* a *AttributeValue ResponseTimeValue* comprising the entire value space (e.g. represented by a subclass of *DOLCE:Temporal-Region*). Moreover, a concept *Preference isAssignedTo ResponseTime* that identifies a *PatternBasedFunction* or *PiecewiseLinearFunction*. These *Functions* map *AttributeValues* to *UtilityValues* as discussed in Section 4.1.2 and 4.1.3.

After presenting how *Configurations* as well as *PolicyDescriptions* are modeled, we introduce the rules for evaluating concrete *Configurations* with respect to given *PolicyDescriptions*. We show how pricing policies are applied to determine the price of a configuration or scoring policies to determine the willingness to pay.

4.2.3 Policy Evaluation

With our approach, policies that define *Preferences* no longer lead only to a pure boolean statement about the conformity of a *Configuration*, but rather to a degree of conformity of the *Configuration*. Therefore, the original *DnS:satisfies*-relation between a *DnS:Situation* and *DnS:SituationDescription* is not sufficient any more since additional information about the degree of conformity has to be captured. However, since checking for satisfaction can be interpreted as the evaluation of the Goal policy aspect in the *PolicyDescription*, meeting this requirement can be seen as a necessary prequisite. That means, if a *Configuration* does not satisfy a *Policy* we can assign a *UtilityValue* of $-\infty$. This is captured by the following rule which refines the *DnS:satsfies*-relation. The reader familiar with DOLCE will notice that Rule (R7) largely corresponds to the *completely-satisfies* relation described in [17]. Since our formalism is not expressive enough to capture this relation directly, we provide a workaround that explicitly enumerates the attributes $1, \ldots, n$ and checks for classification of an appropriate ground entity, thus implementing qualified satisfaction (cf. [17]). Note that we assume an own *AttributeValue* for each attribute.

$$\begin{aligned} \mathit{satisfiesPolicy}(c,p) \leftarrow\ & \mathit{Configuration}(c), \mathit{PolicyDescription}(d), \\ & \mathit{DnS{:}satisfies}(c,p), \bigwedge_{i\in 1\ldots,n} (\mathit{Attribute}_i(a), \\ & \mathit{DnS{:}defines}(p,a_i), \mathit{DnS{:}valuedBy}(a_i,av_i), \\ & \mathit{DnS{:}settingFor}(c,cv_i), \mathit{match}(av_i,cv_i)) \end{aligned} \tag{R7}$$

Ontologically, modeling utility function policies requires putting in relation the *PolicyDescription*, a concrete *Configuration* and an *overallDegree* that represents the valuation to which the latter satisfies the former. For the sake of simplicity and compact representation we use predicates of higher arity in the following. How this could be avoided by introducing an *OIO:InformationObject Satisfiability* that links *Configuration* and *PolicyDescription* is outlined in [34]. If *satisfiesPolicy* does not

hold no further evaluation will be necessary and a value of $-\infty$ is assigned by Rule (R8).

$$overallDegree(c, p, v) \leftarrow \neg satisfiesPolicy(c, p), assign(v, \text{“}-\infty\text{”}) \tag{R8}$$

In line with the additive utility model defined in Equation (1), we first calculate the valuation for each independent set of attributes individually and then aggregate the individual valuations to get the overall degree of a configuration. The local utility values can be calculated by Rules (R1), (R4) and (R6) depending on the type of function used. The valuation derived from these rules can be interpreted as the valuation a single attribute contributes to the overall valuation. Note that the non-additive case is a simplification of this approach, where the local utility value corresponds to the overall value.

$$\begin{aligned} overallDegree(c, p, d) \leftarrow & \bigwedge_{i=1,\ldots,n} (DnS{:}defines(p, a_i), DnS{:}defines(p, pf_i), \\ & isAssignedTo(pf_i, a_i), DnS{:}valuedBy(a_i, av_i), \\ & DnS{:}settingFor(c, cv_i), match(av_i, cv_i), DnS{:}playedBy(pf_i, f_i), \\ & degree(f_i, cv_i, v_i)), sum(v, v_1, \ldots, v_n) \end{aligned} \tag{R9}$$

Rule (R9) is simplified in a sense that predicates for weighting of attributes according to their relative importance λ_i are omitted. However, adding this is straightforward as shown in [34].

To illustrate this approach, we assume a customer with the scoring policies *p* based on the example *Functions* defined in Section 4.1.1 - 4.1.3. We can query the knowledge base to compare the *overallDegree* for *Configuration c* with respect to the *PolicyDescription p*. As an example, we assume a *Configuration* of a route planning service, which returns the cheapest route that includes information about historical sites while considering weather information. Further, a response time of 20 sec. is guaranteed. Evaluating the (local) *degree*-predicates for each *Attribute* leads to a score of 1 for the *Attribute WeatherConsideration*, 0.4 for *RouteType*, 0.8 for *IndicatedAttraction* and 0.47 for *Response Time*, respectively. Provided that all *Attributes* are equally important this *Configuration* results in a *overallDegree* of 0.67.

4.3 Policy Aggregation

Up to now we focused on scenarios where only one policy was used by a single buyer or seller. However, since policy-based approaches are usually applied

in large-scale applications, typically more than one policy may be specified in order to regulate a certain decision. For example, a Web service selection process of a company might be regulated by several scoring policies coming from different departments of the company. The information systems department, for instance, might prefer a highly secure service, while the management might prioritize cheap services. Of course, different scoring policies lead to different valuations as well as rankings and thus to different selections of services. In the remainder of this section, we present a method to derive a coherent decision from such diverse policies. Therefore, policies are first evaluated and the results of this evaluation step are then aggregated.

In traditional policy languages there are two major operators that can be used to combine policies [35,66]: we can use either a logical *and*-operator in order to define a conjunction of policies (i.e. the aggregated policy is admissible if all contained policies are admissible) or a logical *or*-operator to derive a disjunction of policies (i.e. the aggregated policy is admissible if at least one contained policy is admissible).

However, since our policy language results in degrees of satisfiability, this traditional interpretation of the logical operators cannot be used. In order to define the semantics of the logical operators for such multi-valued logics, we borrow ideas from fuzzy logic where the semantics of conjunction and disjunction is defined via *T-norms* and *T-conorms*. In the following, we use the T-norm/T-conorm defined by Zadeh [68] as follows:

$$\top(a,b) = min(a,b) \quad \text{for } \textit{and}\text{-operators} \tag{2}$$

$$\bot(a,b) = max(a,b) \quad \text{for } \textit{or}\text{-operators} \tag{3}$$

We use the definitions above to make sure that if one of the policies is evaluated to $-\infty$, the overall valuation of the conjunction of policies is also $-\infty$. In case of disjunctions only one policy has to be fulfilled and thus we take the maximal valuation.

We next introduce the modeling primitives required for representing conjunctions and disjunctions of policies, as shown in Figure 6. To be able to evaluate a certain *Configuration* with respect to a set of policies, we adapt Rule R9 that it can be used not only for a single *PolicyDescription*, but also for a *PolicyCollection*. A *PolicyCollection* is defined as a *DnS:SituationDescription* that has exactly two *memberPolicy*-relations pointing to *PolicyDescriptions*. This is formalized using the following DL axioms:

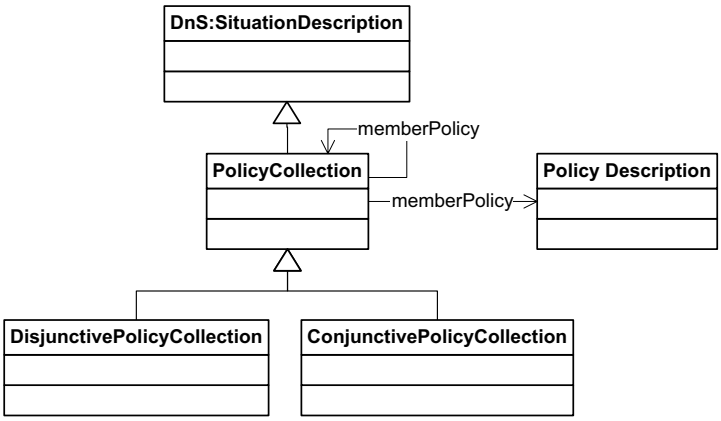

Fig. 6. Policy Collection

$$
\begin{aligned}
\textit{PolicyCollection} \sqsubseteq & \textit{DnS:SituationDescription} \sqcap \\
& =_1 \textit{memberPolicy1}.(\textit{PolicyDescription} \sqcup \\
& \textit{PolicyCollection}) \sqcap \\
& =_1 \textit{memberPolicy2}.(\textit{PolicyDescription} \sqcup \\
& \textit{PolicyCollection}) \qquad \text{(A6)} \\
\textit{memberPolicy1} \sqsubseteq & \textit{DnS:expands} \qquad \text{(A7)} \\
\textit{memberPolicy2} \sqsubseteq & \textit{DnS:expands} \qquad \text{(A8)}
\end{aligned}
$$

The reason why we restrict a *PolicyCollection* to exactly two *memberPolicy*-relations is the fact that SWRL does not support universal quanitifcation in the rule body. Hence, we cannot iterate about an arbitrary number of *PolicyDescriptions* contained in the collection (e.g. the first order logic term '$\forall y.\textit{memberPolicy}(x, y)$' is not expressible in SWRL). However, restricting a *PolicyCollection* to exactly two *memberPolicy*-relations is in fact no limitation, since an arbitrary number of *PolicyCollections* with two *memberPolicy*-relations can be nested which has the same effect as multiple *memberPolicy*-relations within one *PolicyCollection*.

In order to define a relation between the members of a *PolicyCollection* we introduce the two subclasses of *PolicyCollection*, *ConjunctivePolicyCollection* and *DisjunctivePolicyCollection*. Then, for each of these subclasses a rule is introduced that calculates the *overallDegree* of the collection based on the *overallDegrees* of the elements contained. The following rule does the calculation for a *ConjunctivePolicyCollection* where the individual elements are connected by a logical *and*-relation based on the T-norm defined in Equation (2).

$$overallDegree(c, p, v) \leftarrow ConjunctivePolicyCollection(p),$$
$$\bigwedge_{i\in\{1,2\}} (memberPolicy_i(p, p_i), overallDegree(c, p_i, v_i)),$$
$$min(v, v_1, v_2) \qquad \text{(R10)}$$

Note that Rule (R10) recursively calculates the *overallDegree* of the elements contained in the collection. Rule (R10) will only be used if a *ConjunctivePolicyCollection* is passed to the *overallDegree*-predicate. If it refers to a single *PolicyDescription*, Rule (R9) will be applied as before.

Analogously, we can define the Rule (R11) for *DisjunctivePolicyCollections* where the T-conorm (Equation (3)) is used to calculate the *overallDegree*.

$$overallDegree(c, p, v) \leftarrow DisjunctivePolicyCollection(p),$$
$$\bigwedge_{i\in\{1,2\}} (memberPolicy_i(p, p_i), overallDegree(c, p_i, v_i)),$$
$$max(v, v_1, v_2) \qquad \text{(R11)}$$

DisjunctivePolicyCollections and *ConjunctivePolicyCollections* can be nested within each other provided that the leafs of the emerging tree structure are always primitive *PolicyDescriptions*.

5 Core Ontology of Bids

After having introduced a policy ontology for specifying valuation functions over multi-attribute objects or activities, we show how such policies are used for attaching price information to goods or services in the following. As introduced in Section 1, a statement that captures such information is called a *bid* in economic literature. For modeling bids we apply the pattern Description & Situation again. In line with the structure of the previous section, we first define how to specify *trades* in Section 5.1. Trades capture one possible transaction in the market and define exactly the objects and services to be exchanged and their concrete configuration. Based on this definition we introduce the specification of *Bids*, which can be seen as views that select the desired subset of trades (Section 5.2). Finally in Section 5.3, a method for evaluating bids is presented.

5.1 Specification of Trades

As defined in [14], a (bilateral) *trade* is a potential transaction between two parties *b* and *s* where agent *b* buys an object or service from agent *s* for a certain amount of money π. We model this by introducing a special *DnS:Situation* called TradeSituation which extends the *CPO:Configuration* (Axiom (A11)). In this context, we restrict trades to two classes of products: the class of *Goods* as specialization of *DnS:Endurant* and the class *Service* as specialization of *DnS:Activity* (Axiom (A9) and (A10)). Since these *Services* or *Goods* are multi-attributive they have to refer to a *CPO:Configuration* that defines the values of the different properties of these products, as discussed in Section 4.2.1. A certain *TradeSituation* should refer to exactly one *CPO:Configuration* and could specify the corresponding price π which is modeled via the *DOLCE:Region PriceValue*. Moreover, at least one *DnS:Agent* has to be part of the *TradeSituation* (Axiom (A11)). Note that this formalization does not require to specify both participants – *b* and *s* – of a trade, since this is usually not needed in the bid evaluation process.

$$Service \sqsubseteq DnS{:}Activity \sqcap \exists DnS{:}definedBy.CPO{:}Configuration \tag{A9}$$

$$Good \sqsubseteq DnS{:}Endurant \sqcap \exists DnS{:}definedBy.CPO{:}Configuration \tag{A10}$$

$$TradeSituation \sqsubseteq DnS{:}Situation \sqcap \exists DnS{:}settingFor.(Service \sqcup Good) \sqcap =_1 DOLCE{:}part.CPO{:}Configuration \sqcap \exists DnS{:}settingFor.DnS{:}Agent \tag{A11}$$

$$PriceValue \sqsubseteq DOLCE{:}Region \tag{A12}$$

The lower part of Figure 7 illustrates the specification of a *TradeSituation* – called *John'sTrade* – by means of an example: *John* provides the route planning service *John'sService* to a price of $2 per invocation. Moreover, John provides a certain configuration *Conf1*. The specification of *Conf1* is omitted in Figure 7, since an example for modeling *Configurations* is already given in Section 4. Since the price is explicitly modeled as a property of the service, for each additional configuration John wants to provide, a new *TradeSituation* instance has to be introduced. Therefore, enumeration based approaches are only feasible for very low number of configurations.[7] In order to avoid such enumerations the concept of *Bid* is presented in the following section.

[7] In addition, this approach is imprecise from an ontological point of view, since a price is not an inherent quality of a product that can be observed but might depend on the context and other factors.

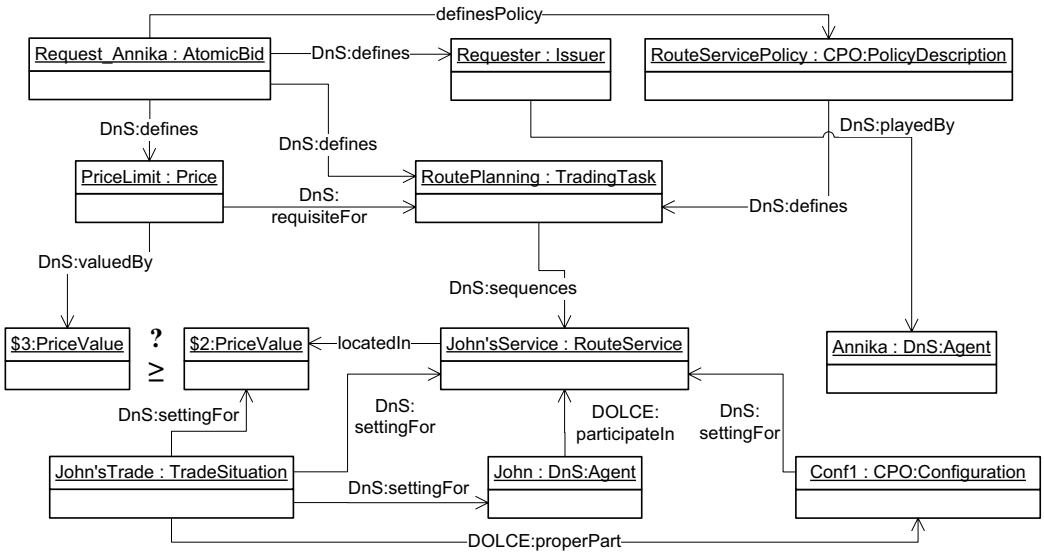

Fig. 7. Example for a *TradingSituation* and *AtomicBid*. The parts of the diagram are successively introduced in Sections 5.1–5.3.

5.2 Specification of Bids

Not all trades that are possible in a market are favorable for an agent. According to Engel et al. [14], a bid expresses the willingness to participate in trades. We thus model a bid as *DnS:Description* that *DnS:classifies* exactly those *TradeSituations* in which the *Issuer* of a bid is willing to participate. In such *BidDescriptions Goods* and *Services* of a concrete *TradeSituation* play the role of *TradingObjects* and *TradingTask*, respectively. In order to implement matching in the market one has to define what entities can be *DnS:classifiedBy* a *TradingObject* or *TradingTask*, e.g. that a *RoutePlanningTasks DnS:sequences* only *RoutePlanningServices*. Moreover, the description defines a Parameter *Price* that constrains these *TradingObjects* and *TradingTasks*. This *Price* can be defined explicitly for each service configuration or implicitly by means of *CPO:PolicyDescriptions*. In our ontology we capture this by introducing the concept *AtomicBid* as follows:

$$\begin{aligned} \mathit{AtomicBid} \sqsubseteq\ & \mathit{DnS{:}SituationDescription} \sqcap \exists \mathit{DnS{:}defines}.(\mathit{TradingObject} \sqcup \\ & \mathit{TradingTask}) \sqcap =_1 \mathit{DnS{:}defines.Price} \sqcap \\ & \forall \mathit{definesPolicy}.(\mathit{CPO{:}PolicyDescription} \sqcap \\ & \mathit{CPO{:}PolicyCollection}) \qquad \text{(A13)} \\ \mathit{definesPolicy} \sqsubseteq\ & \mathit{DnS{:}expandedBy} \qquad \text{(A14)} \\ \mathit{TradingObject} \sqsubseteq\ & \mathit{CPO{:}PolicyObject} \sqcap \\ & \exists \mathit{DnS{:}playedBy.DOLCE{:}Endurant} \qquad \text{(A15)} \\ \mathit{TradingTask} \sqsubseteq\ & \mathit{CPO{:}PolicyTask} \sqcap \\ & \exists \mathit{DnS{:}sequences.DOLCE{:}Perdurant} \qquad \text{(A16)} \end{aligned}$$

$$Price \sqsubseteq DnS{:}Parameter \sqcap =_1 DnS{:}requisiteFor.(TradingObject \sqcup TradingTask) \sqcap \forall DnS{:}valuedBy.PriceValue \quad \text{(A17)}$$

$$Issuer \sqsubseteq DnS{:}Role \sqcap \exists DnS{:}playedBy.DnS{:}Agent \quad \text{(A18)}$$

In addition, a *Price* might represent a maximal price (*MaxPrice*) or a minimal price (*minPrice*). As formalized in Axiom (A19) and (A20), we denote an *AtomicBid* with minimal price as *Offer* and an *AtomicBid* with maximal price as *Request*. That means *Offers* classify *TradeSituations* where the property *PriceValue* is above a certain threshold and *Requests* classify *TradeSituations* where *PriceValue* is below the threshold.

$$Offer \sqsubseteq AtomicBid \sqcap \exists DnS{:}defines.MinPrice \quad \text{(A19)}$$

$$Request \sqsubseteq AtomicBid \sqcap \exists DnS{:}defines.MaxPrice \quad \text{(A20)}$$

Some market mechanisms support more complex bid specifications beyond the simple case of *AtomicBids* [44]. Most prominent in this context are combinatorial bids that enable expressing superadditive as well as subadditive prices for a bundle of products. Superadditive prices occur in case of complementary products that are usually used together, such as desktop computers and computer monitors. For such products the value of a bundle containing both products is typically valued higher by a customer than the sum of the value for the single products. Similarly subadditivity describes substitutes where products suit the same purpose, e.g. a laptop and a desktop computer. In line with [55] we model superadditivity by introducing *ANDBids* and subadditivity by means of *XORBids*. Intuitively, *ANDBids* are bids on several products where one would like to have all of them or nothing. In case of *XORBids* exactly one product should be allocated. As formalized in Axiom (A23) and (A24), *ANDBids* and *XORBids* are specialization of *BundleBid* which are all *Bids* that consist of exactly two other *Bids* (Axiom (A21)). A *Bid* represents the superconcept of *AtomicBids* and *BundleBids*. While *ANDBids* have to contain a *Price* attached to each bundle (either explicitly or using policies), no *Prices* can be attached to *XORBids*, since in this context only the *Prices* of the *AtomicBids* are relevant. Note that since each *BundleBid* has to contain exactly two *Bids*, all *BundleBids* have to terminate solely with *AtomicBids* in a consistent knowledge base (possibly after an arbitrary number of nested *BundleBids*). The axioms below formalize combinatorial bids.

$$BundleBid \sqsubseteq DnS{:}SituationDescription \sqcap =_2 consistsOf.Bid \quad \text{(A21)}$$

$$consistsOf \sqsubseteq DnS{:}expandedBy \quad \text{(A22)}$$

$$\begin{aligned} \mathit{ANDBid} \sqsubseteq & \mathit{BundleBid} \sqcap =_1 \mathit{andRelated1}.(\mathit{AtomicBid} \sqcup \mathit{ANDBid}) \sqcap \\ & =_1 \mathit{andRelated2}.(\mathit{AtomicBid} \sqcup \mathit{ANDBid}) \sqcap \\ & \exists \mathit{DnS{:}defines.Price} \sqcap \forall \mathit{definesPolicy}.(\mathit{CPO{:}PolicyDescription} \sqcap \\ & \mathit{CPO{:}PolicyCollectinon}) \end{aligned} \tag{A23}$$

$$\mathit{XORBid} \sqsubseteq \mathit{BundleBid} \sqcap =_1 \mathit{xorRelated1.Bid} \sqcap =_1 \mathit{xorRelated2.Bid} \tag{A24}$$

$$\mathit{Bid} \equiv \mathit{AtomicBid} \sqcup \mathit{BundleBid} \tag{A25}$$

The relations *andRelated1* and *andRelated2* as well as *xorRelated1* and *xorRelated2* are all modeled as subproperties of *consistsOf*. As done in Section 4.3 for the concept *PolicyCollection*, we fix the number of *Bids* in a *BundleBid* by explicitly introducing two *consistsOf*-relations. This technique allows us to avoid universal quantification in rule bodies which is not supported by our rule language. Due to the fact that bundles can be nested, an arbitrary number of *AtomicBids* can be combined.

A simple *AtomicBid* is exemplified in the upper part of Figure 7. Annika needs a service for a route planning task. Therefore, she instantiates *AtomicBid* and *DnS:defines* a *TradingTask* called *RoutePlanning*. Her willingness to pay is specified implicitly via her policy *RouteServicePolicy*. That means the *DnS:Parameter Price* is *DnS:valuedBy* a *PriceValue* that has to be calculated with respect to a concrete *TradeSituation*. This evaluation of a bid is discussed in Section 5.3.

5.3 Bid evaluation

Bids are *DnS:Descriptions* that select *TradeSituations* that fulfill the specified requirements. Requirements are expressed via *CPO:PolicyDescriptions*. Therefore, evaluation of *Bids* can be largely reduced to policy evaluation. Rule (R12) determines the *PriceValue p* of a *AtomicBid b* with respect to a concrete *TradeSituation t* using the predicate *overallDegree* which has been introduced in Section 4.2.3.

$$\begin{aligned} \mathit{price}(b,t,p) \leftarrow & \mathit{AtomicBid}(b), \mathit{CPO{:}PolicyDescription}(d), \\ & \mathit{definesPolicy}(b,d), \mathit{TradeSituation}(t), \mathit{DnS{:}settingFor}(t,c), \\ & \mathit{CPO{:}Configuration}(c), \mathit{overallDegree}(c,d,p) \end{aligned} \tag{R12}$$

For *BundleBids* we apply Rule (R12) for each *AtomicBid* contained in the bundle. In case of *XORBids* only one *Bid* in the bundle has to be fulfilled. We thus evaluate the *TradeSituation* with each contained *Bid* separately and then determine the price of the *AtomicBid* that is most suitable. Rule (R13) captures this in a recursive manner.

$$price(b,t,p) \leftarrow XORBid(b), xorRelated1(b,b1), price(b1,t,p1), \\ xorRelated2(b,b2), price(b2,t,p2), swrlb{:}max(p,p1,p2) \quad \text{(R13)}$$

For calculating the price of an *ANDBid* only policies attached to the *ANDBid* itself are considered. This is realized simply by replacing the term *AtomicBid*(*b*) in Rule (R12) with *ANDBid*(*b*).

After introducing the calculation of a *Bid*'s *PriceValue* we can define the *satisfies-Bid*-relation that determines if a certain *TradeSituation* is acceptable according to a *Bid*. For the case where *Services* are traded, the following rule checks whether the right service is provided in the *TradeSituation* and whether the price is in an acceptable range (which is defined by the policy). For comparing the TradingTask we use the built-in *subsumes*, which has already been used in Rule (R3). Thereby, we make sure that the provided *Service* fulfills the same purpose as the *Service* sequenced by the *TradingTask*. As already discussed in Section 4.1.1, a meta-modeling approach is required where *Services* are seen as concepts as well as individuals.

$$satisfiesBid(b,t) \leftarrow Bid(b), TradeSituation(t), DnS{:}defines(b,o), \\ TradingTask(o), DnS{:}sequences(o,e), DnS{:}settingFor(t,d), \\ DOLCE{:}Service(d), type(e,d), DnS{:}defines(b,t,pb), \\ MaxPrice(pb), DnS{:}settingFor(t,pt), \\ swrlb{:}lessThanOrEqual(pt,pb) \quad \text{(R14)}$$

Rule for *Offers* and for *Bids* containing *TradingObjects* are defined analogously. To illustrate this approach, we come back to the example in Figure 7. Here we are interested if the *TradeSituation John'sTrade* is relevant for Annika's bid (*Request_Annika*). In order to determine the maximal price Annika is willing to pay for the *Configuration Conf1* provided by John, we use Rule (R12). Assume the result of this evaluation step is a *MaxPrice* of \$3. For checking if John provides the right service, we assume the following definition: $RoutePlanning \sqsubseteq OoP{:}Task \sqcap \forall DnS{:}sequences.RouteService$. Since John provides exactly this type of service for \$2, the *subsumes*-predicate as well as the *lessThanOrEqual*-predicate evaluate to true and the *TradeSituation* satisfies the *Bid*. Note that if John defines the service price also via policies, the evaluation leads to a more complex optimization problem which is beyond the scope of this paper. The interested reader is referred to [33] for a more detailed discussion.

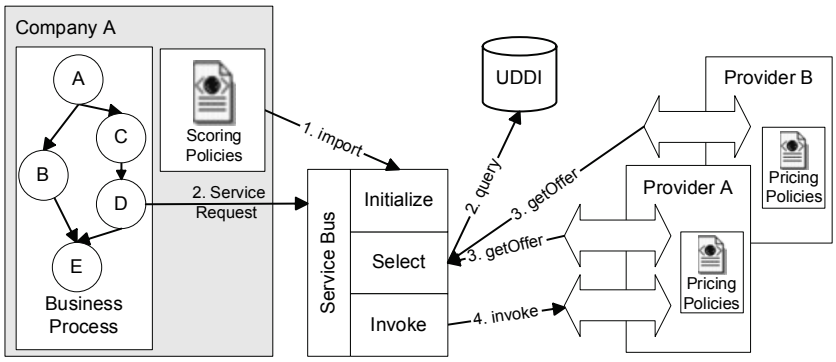

Fig. 8. Service Bus Architecture

6 Proof of Concept

After having introduced an ontology framework for a compact representation of multi-attribute as well as configurable bids, we present here an application of the framework as proof of concept. We introduce the scenario of dynamic Web service selection in Section 6.1 and in Section 6.2 a prototype that implements our framework in this setting. Finally, in Section 6.3 we evaluate the bid specification as well as the bid evaluation using this prototype.

6.1 Service Bus

A *service bus* architecture enables dynamic Web service selection and avoids hard-wiring of Web services within a service-oriented implementation of a business process. Dynamic selection of services is required in service oriented architectures, because often at development time of the business process the concrete Web service for a certain task is not known (e.g. the choice might depend on runtime specific context). In addition, the set of services available for accomplishing a certain task may change frequently. In addition, dynamic selection provides a high flexibility of the implementation since switching from one service to another can be done automatically during run-time without changing code. This can lead to more robust systems and to lower costs, since erroneous and expensive services can be automatically replaced.

Figure 8 exemplifies the architecture of service bus architecture. On the left side, a company's business process is visualized as a workflow of tasks that have to be accomplished by a Web service, e.g. calculation of a route. The company further defines general policies how the business process should behave. These policies have to contain scoring policies specifying the company's preferences about Web service properties. Once a Web service is required within the business process (step D), the following steps have to be carried out:

(1) Together with the first service request, scoring policies including attribute weighting information are sent to the service bus and stored in a knowledge base. Note that this initialization step only has to happen once for the initialization of the system. Based on this scoring policy, the service bus is able to take over the responsibility of selecting between potential providers, such as A and B. In this sense a service bus can be seen as a simple take-it or leave-it market for Web services. Since the decision for a certain service might depend on runtime information, different scoring policies might have to be specified for different contexts (current location, time, etc.).
(2) Once a request from an application arrives, the service bus first queries a UDDI registry for suitable providers. Here only a very simple matching of the service functionality is carried out. This means only the addresses of services are returned that provide the required functionality.
(3) In a next step, offers from the providers are collected in parallel. These offers contain a list of provided configurations together with the pricing policies of the corresponding provider. The pricing policies are also stored in the knowledge base of the service bus.
(4) Finally, the service bus queries the knowledge base for all service offers and configurations that fulfill the required functionality. A list of services ranked according to the difference between score and price is returned. Based on the ranking, the best provider is selected and the respective service invoked. In case this invocation fails, the second best service is chosen. This is repeated until the required task is accomplished or no acceptable service remains.

In the next section we present a prototype that (partially) implements this functionality based on the ontology framework introduced above.

6.2 Prototype

Our prototype [8] consists of two components: (i) A server component provides a repository for Web service offers and thus implements the Service Bus described in the previous section. The repository is a DL knowledge base that can be queried using the KAON2 inference engine. [9] KAON2 is chosen because it supports the description logic $\mathcal{SHIQ}$ as well as DL-safe rules and thus the ontology as well as rule language required for our bid descriptions. KAON2 has been optimized for query answering [41], which is the required functionality for bid evaluation. In addition, due to the lack of a publicly available UDDI repository we added components to the server that crawl the Web, collect WSDL files and HTML forms and transform them to offer descriptions. (ii) The second component is a client that facilitates

[8] More information about the prototype is available at <`http://kasws.sourceforge.net/`> (accessed 17.03.2008).
[9] Available at <`http://kaon2.semanticweb.org/`> (accessed 17.03.2008).

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 [22]. Generally, the framework supports more expressive service descriptions than discussed in this paper. For example, the service description could include behavioral aspects as presented by Agarwal and Studer [1].

Coming back to our initial scenario, the prototype allows Company A in Figure 8 to specify a *Request* and the query for a concrete service. To query the KAON2 repository the query is formalized using SPARQL [67]. It can be used to derive all suitable *Services* from the knowledge base ranked according to the scoring policies specified by the company. To illustrate this approach, the following query is used to find all *Services* that satisfy a certain *Bid b* and to rank these *Services* according to difference between willingness to pay specified in the request and the actual price.

```
   BASE  < http://emo.ontoware.org/0.1/ >
 SELECT  ?service ?utility
 WHERE { b  satisfiesBid  ?trade .
         EVALUATE ?bidPrice := price(b,?trade) .
          ?trade  DnS:settingFor  ?price ; ?service .
          ?service  rdf:type  Service .
          ?price  rdf:type  PriceValue .
         EVALUATE ?utility := sub(?bidPrice,?price) . }
ORDER BY DESC(?utility)
```

Instead of taking care of the business process execution itself, the client tool can be augmented with a WS-BPEL engine [47]. To enable dynamic service selection at runtime, Web service queries generated with the client tool can serve as abstract goal/task descriptions in a BPEL4WS process. At execution time these descriptions are used to query the repository and the address of the selected service is dynamically assigned to corresponding invoke-statement. However, dynamic reassignment of ports in BPEL4WS is only feasible if alternative services have identical interfaces, i.e. WSDL port types [52]. Scenarios beyond this simple case are currently not covered by our prototype.

6.3 Evaluation

Our work presents a novel approach to express preferences of market participants that relies on the concept of utility function policies. In this section, we discuss the consequences of such a modeling approach on communication efficiency and

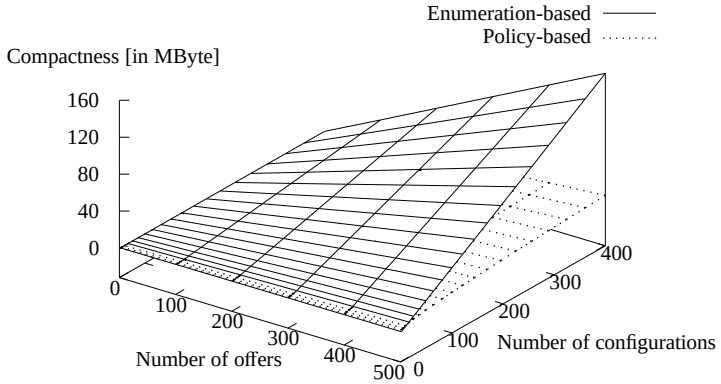

Fig. 9. Number of axioms in KB.

bid evaluation complexity. Thus, this section addresses mainly two aspects: First, in terms of bid specification we are interested whether our functional modeling approach reduces the storage capacity and therefore improves communication efficiency in the market. Second, we are interested in the performance of the bid evaluation process. For both aspects we compare the policy-based bid specification with a baseline approach that enumerates all provided or requested configurations.

To investigate these questions, we have randomly generated Web service offers with a varying number of configurations using a uniform distribution and stored them in the repository. For one set of offers prices are defined implicitly via policies, whereas for the other set price information is explicitly attached to each configuration. Figure 9 shows the number of axioms that are required to represent multi-attribute offers in the market. The number of possible configurations per offer is increased from 1 to 400 and the number of offers published in the repository from 1 to 500. Already with two configurations the policy-based approach requires less axioms than the enumeration-based approach for expressing the same information. If we further increase the number of configurations per offer, the number of axioms required for the enumeration-based approach increases linearly, which leads to an overall space complexity of $O(|O||C|)$. Because the set C of configurations is defined as $C = \prod_j A_j$ (compare Section 3.1), the number of configurations grows exponentially with the number of attributes. We thus get a space complexity of $O(|O|n^{|A|})$, where n represents the maximal number of attribute values of an attribute ($n = \max_j A_j$). For the policy-based approach, in contrast, axioms increase linearly with the number of attributes $\sum_j |A_j|$ and thus this approach exhibits a logarithmic space complexity with respect to the number of configurations. In addition, no *PriceValue* instance for each *Offes* and *Configuration* instance has to be introduced. The policy-based approach leads therefore to an overall space complexity of $O(|O| + log(|C|))$ and $O(|O| + n|A|)$, respectively. Note that this holds only for discrete attributes. Continuous attributes can be specified even more efficiently with the policy-based approach. A representation using enumeration is not possible at

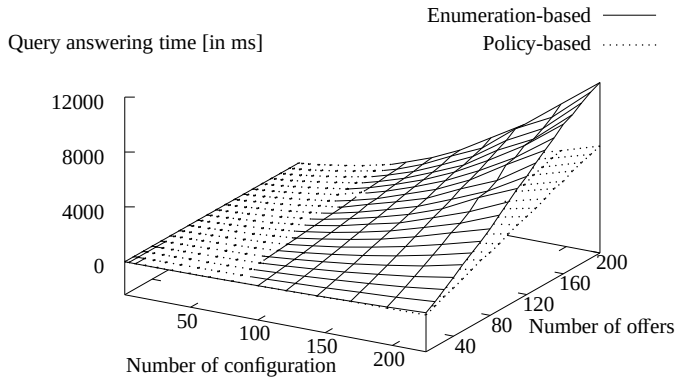

Fig. 10. Query answering time.

all. In fact, only a constant number of axioms might be required for continuous attributes and a space complexity of $O(|O|)$ can be reached. Therefore, we can conclude that policy-based bid specification considerably improves scalability in the presence of highly configurable products.

In the second step, we investigate the performance of bid evaluation. Therefore, we send queries to the repository and measure the time that elapses until we receive the ranked list of configurations and offers. To avoid network delays this simulation is performed on a single computer. Figure 10 shows the average query answering time depending on the number of configurations and offers. Since the time complexity of query answering is completely predefined by the ontology language as well as the corresponding reasoning algorithms, theoretically no difference between the complexity of the two approaches exists. However, in practice some differences can be observed.

As depicted in Figure 10, while query answering in case of enumeration is extremely fast for small scenarios (i.e. less than 60 configurations), the lookup time of the *PriceValue* instances in the knowledge base increases considerably with an increasing size of the A-box (indicated by the solid lines in Figure 10). For example, the service selection with 200 offers each referring to 20 configurations can be done within 194ms, whereas query answering with 200 offers and 225 configurations requires already 10 sec. If we further increase the complexity to 800 offers and 600 configurations the enumeration-based approach takes 6 minutes for selecting the best service. This is clearly too long for service election at runtime.

In case of policy-based descriptions similar performance characteristics can be identified, however, on a lower level (indicated by the dashed lines in Figure 10). In small scenarios the policy-based approach is outperformed by the enumeration-based approach, while for medium and large scenarios the policy-based approach performs considerably better. Assuming 200 offers each with 20 configurations the selection can be done in 301ms, whereas in the case of 800 offers and 600 configu-

rations the selection requires 2 minutes. This is an improvement of over 60% compared to the enumeration-based approach. As a reason for the poor scalability of the enumeration-based approach the exponential increase of *PriceValue* instances can be identified. This exponential explosion can be avoided using the policy-based approach. However, in small scenarios the evaluation of Rules (R12) and (R13) is more expensive than looking up the right price in the knowledge base. As discussed in [32], this issue can be addressed with intelligent caching algorithms that, e.g., explicitly store price information of frequently queried configurations. Using such a strategy, the costly policy evaluations have to be done only once for the first query.

Although the policy-based approach does provide a considerably improved scalability compared to the enumeration-based approach, it might still not be sufficient for dynamic service selection at runtime. Therefore, in [33] two alternative matching variants have been introduced. However, they are only applicable in scenarios where the no full ranking of all configurations is required and where preferences have an additive structure. They utilize a linear programming formulation and standard linear programming solvers. Using such techniques a considerably speedup of the policy-based approach can be realized. For example, for the setting of 800 offers and 600 configurations an improvement of more than 90% can be observed.

7 Conclusion

In this paper, we have provided an ontology framework that captures multi-attribute combinatorial requests and offers. In this context, we have presented the *Core Policy Ontology* that realizes the advantages of utility function policies, such as preference modeling and inherent conflict resolution, with a purely declarative, Web compliant and standard-based approach. In addition, we introduced the *Core Ontology of Bids* that uses utility function policies for compact representation of bids for configurable goods and services. The specification and evaluation mechanism we have presented is particularly tailored for Web-based markets by providing flexibility and interoperability in heterogenous environments. In order to exemplify our approach we have developed a prototype which enables dynamic selection of Web services. Since Web technologies facilitate customization and personalization of products, we believe that expressing offers and requests for configurable products in a compact and interoperable way will be crucial for future markets on the Web.

In future, we plan to extend our approach in several directions. First, we are going to investigate how matching of offers and requests can be realized purely based on policies without explicitly considering each possible trade. Second, we plan to extend the Core Policy Ontology in a way that pricing and scoring policies can be defined on behavioral aspects of services (cf. [1,10]). Third, in terms of implementation we plan to enable dynamic service selection beyond simple port reassignments.

Acknowledgements

Research reported in this paper has been financed by the German Research Foundation (DFG) in scope of the Graduate School for Information Management and Market Engineering (DFG grant no. GRK 895), the SmartWeb project of the Federal Ministry of Education and Research (BMBF), and the EU in the IST projects SUPER (FP6-026850) and GENESIS (FP6-027867). In addition, we thank our colleague Denny Vrandečić and the reviewers and participants of the ICEC'06 conference for their valuable comments and suggestions.

References

[1] S. Agarwal, R. Studer, Automatic matchmaking of web services, in: Proceedings of IEEE International Conference on Web Services (ICWS 2006), Chicago, Illinois, USA, IEEE Computer Society, Los Alamitos, CA, USA, 2006, pp. 45–54.

[2] American National Standards Institute (ANSI), ANSI ASC X12, Release 5040 (January 2007). <http://www.x12.org/> (accessed 04.03.2008).

[3] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Kakata, J. Pruyne, J. Rofrano, S. Tuecke, M. Xu, Web services agreement specification (WS-Agreement), GFD.107 (May 2007). <http://www.ogf.org/documents/GFD.107.pdf> (accessed 04.03.2008).

[4] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. F. Patel-Schneider (eds.), The Description Logic Handbook: Theory, Implemenation and Applications, Cambridge University Press, 2003.

[5] F. Bacchus, A. Grove, Graphical models for preference and utility, in: P. Besnard, S. Hanks (eds.), Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence, Montreal, Quebec, Canada, Morgan Kaufmann, 1995, p. 310.

[6] Y. Bakos, The emerging role of electronic marketplaces on the internet, Communications of the ACM 41 (8) (1998) 35–42.

[7] T. Berners-Lee, J. Hendler, O. Lassila, The semantic web, Scientific American 284 (5) (2001) 34–43.

[8] A. Bernstein, C. Kiefer, Imprecise RDQL: Towards generic retrieval in ontologies using similarity joins, in: H. Haddad (ed.), Proceedings of the 21th Annual ACM Symposium on Applied Computing, Dijon, France, ACM Press, New York, NY, USA, 2006, pp. 1684–1689.

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

[10] M. Bienvenu, C. Fritz, S. A. McIlraith, Planning with qualitative temporal preferences, in: P. Doherty, J. Mylopoulos, C. A. Welty (eds.), Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, AAAI Press, 2006, pp. 134–144.

[11] C. Boutilier, H. H. Hoos, Bidding languages for combinatorial auctions, in: B. Nebel (ed.), Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, Seattle, Washington, USA, Morgan Kaufmann, 2001, pp. 1211–1217.

[12] S. Brockmans, R. Volz, A. Eberhart, P. Löffler, Visual modeling of OWL DL ontologies using UML, in: S. A. McIlraith, D. Plexousakis, F. van Harmelen (eds.), Proceedings of the Third International Semantic Web Conference (ISWC), Hiroshima, Japan, vol. 3298 of Lecture Notes in Computer Science, Springer, 2004, pp. 198–213.

[13] L. Chen, P. Pu, Survey of Preference Elicitation Methods, Tech. rep., Ecole Politechnique Federale de Lausanne (EPFL), IC/2004/67 (2004). <http://infoscience.epfl.ch/getfile.py?recid=52659> (accessed 04.03.2008).

[14] Y. Engel, M. P. Wellman, K. M. Lochner, Bid expressiveness and clearing algorithms in multiattribute double auctions, in: EC '06: Proceedings of the 7th ACM conference on Electronic commerce, Ann Arbor, Michigan, USA, ACM Press, New York, NY, USA, 2006, pp. 110–119.

[15] European Union, Common procurement vocabulary, adopted by regulation (EC) no. 2195/2002 (December 2003). <http://simap.eu.int/codes-and-nomenclatures/codes-cpv_en.html> (accessed 14.03.2008).

[16] P. C. Fishburn, Utility Theory for Decision Making., Wiley, NewYork, 1970.

[17] A. Gangemi, S. Borgo, C. Catenacci, J. Lehmann, Task taxonomies for knowledge content, Deliverable D07, EU 6FP METOKIS Project (June 2004). <http://metokis.salzburgresearch.at> (accessed 04.03.2008).

[18] A. Gangemi, M.-T. Sagri, D. Tiscornia, A constructive framework for legal ontologies, in: R. Benjamins, P. Casanovas, J. Breuker, A. Gangemi (eds.), Law and the Semantic Web: Legal Ontologies, Methodologies, Legal Information Retrieval, and Applications, Springer, 2005.

[19] S. Grimm, B. Motik, C. Preist, Variance in e-business service discovery, in: D. Martin, R. Lara, T. Yamaguchi (eds.), Proceedings of the ISWC 2004 Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications, Hiroshima, Japan, CEUR Workshop Proceedings, 2004.

[20] B. Grosof, T. Poon, Sweetdeal: Representing agent contracts with exceptions using XML rules, ontologies, and process descriptions, in: Proceedings of the 12th World Wide Web Conference, Budapest, Hungary, ACM Press, New York, NY, USA, 2003, pp. 340–349.

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

[22] P. Haase, B. Motik, A mapping system for the integration of OWL-DL ontologies, in: A. Hahn, S. Abels, L. Haak (eds.), IHIS 05: Proceedings of the First International Workshop on Interoperability of Heterogeneous Information Systems, Bremen, Germany, ACM Press, New York, NY, USA, 2005, pp. 9–16.

[23] I. Horrocks, P. F. Patel-Schneider, A proposal for an OWL rules language, in: WWW '04: Proceedings of the 13th international conference on World Wide Web, New York, NY, USA, ACM Press, New York, NY, USA, 2004, pp. 723–731.

[24] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, SWRL: A semantic web rule language combining OWL and RuleML, W3C Submission (May 2004). <http://www.w3.org/Submission/SWRL> (accessed 07.03.2008).

[25] I. Horrocks, P. F. Patel-Schneider, F. van Harmelen, From SHIQ and RDF to OWL: The making of a web ontology language, Journal of Web Semantics 1 (1) (2003) 7–26.

[26] I. Horrocks, U. Sattler, S. Tessaris, S. Tobies, How to decide query containment under constraints using a description logic, in: Proceedings of the 7th International Conference on Logic for Programming and Automated Reasoning (LPAR'2000), Reunion Island, France, 2000, pp. 326–343.

[27] IBM Corporation, Enterprise privacy authorization language (EPAL 1.2), W3C Member Submission (November 2003). <http://www.w3.org/Submission/2003/SUBM-EPAL-20031110/> (accessed 07.03.2008).

[28] L. Kagal, A Policy-Based Approach to Governing Autonomous Behavior in Distributed Environments, Ph.D. thesis, University of Maryland Baltimore County, Baltimore MD 21250 (November 2004).

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

[30] J. O. Kephart, W. E. Walsh, An artificial intelligence perspective on autonomic computing policies, in: Proceedings of 5th IEEE International Workshop on Policies for Distributed Systems and Networks, Yorktown Heights, New York, USA, 2004, pp. 3–12.

[31] M. Klusch, B. Fries, K. Sycara, Automated semantic web service discovery with OWLS-MX, in: H. Nakashima, M. P. Wellman, G. Weiss, P. Stone (eds.), AAMAS '06: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Aystems, Hakodate, Japan, ACM Press, New York, NY, USA, 2006, pp. 915–922.

[32] S. Lamparter, A. Ankolekar, Automated selection of configurable web services, in: 8. Internationale Tagung Wirtschaftsinformatik (WI2007), Karlsruhe, Germany, Universitätsverlag Karlsruhe, 2007, pp. 441–460.

[33] S. Lamparter, A. Ankolekar, S. Grimm, R. Studer, Preference-based selection of highly configurable web services, in: C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, P. J. Shenoy (eds.), Proc. of the 16th Int. World Wide Web Conference (WWW'07), Banff, Canada, ACM Press, New York, NY, USA, 2007, pp. 1013–1022.

[34] S. Lamparter, A. Ankolekar, D. Oberle, R. Studer, C. Weinhardt, A policy framework for trading configurable goods and services in open electronic markets, in: B. Spencer, M. S. Fox, W. Du, D. Du, S. Buffett (eds.), Proceedings of the 8th International Conference on Electronic Commerce (ICEC'06), New Brunswick, Frederiction, Canada, ACM Press, New York, NY, USA, 2006, pp. 162–173.

[35] S. Lamparter, A. Eberhart, D. Oberle, Approximating service utility from policies and value function patterns, in: 6th IEEE Int. Workshop on Policies for Distributed Systems and Networks, Stockholm, Sweden, IEEE Computer Society, Los Alamitos, CA, USA, 2005, pp. 159–168.

[36] L. Li, 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, Budapest, Hungary, ACM Press, New York, NY, USA, 2003, pp. 331–339.

[37] T. W. Malone, K. Crowston, B. P. J. Lee, C. Dellarocas, G. Wyner, J. Quimby, C. S. Osborn, A. Bernstein, G. Herman, M. Klein, E. O'Donnell, Tools for inventing organizations: Toward a handbook of organizational processes, Management Science 45 (3) (1999) 425–443.

[38] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, A. Oltramari, L. Schneider, The WonderWeb library of foundational ontologies, WonderWeb Deliverable D17 (August 2002). <http://wonderweb.semanticweb.org> (accessed 07.03.2008).

[39] T. Moses, A. Anderson, S. Proctor, S. Godik, XACML profile for web services, Oasis Working Draft (September 2003). <http://www.oasis-open.org/committees/download.php/3661/draft-xacml-wspl-04.pdf> (accessed 07.03.2008).

[40] B. Motik, On the properties of metamodeling in OWL, in: Y. Gil, E. Motta, R. Benjamins, M. A. Musen (eds.), Proceedings of the 4th International Semantic Web Conference (ISWC 2005), Galway, Ireland, vol. 3729 of Lecture Notes in Computer Science, Springer, 2005, pp. 548–562.

[41] B. Motik, U. Sattler, A comparison of reasoning techniques for querying large description logic aboxes, in: M. Hermann, A. Voronkov (eds.), Proceedings of the 13th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2006), Phnom Penh, Cambodia, vol. 4246 of Lecture Notes in Computer Science, Springer, 2006, pp. 227–241.

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

[43] D. Neumann, Market engineering - a structured design process for electronic markets, Ph.D. thesis, Department of Economics and Business Engineering, University of Karlsruhe (TH), Karlsruhe (2004).

[44] N. Nisan, Bidding and allocation in combinatorial auctions, in: Proceedings of the 2nd ACM Conference on Electronic Commerce (EC'00), Minneapolis, MN, USA, ACM Press, New York, NY, USA, 2000, pp. 1–12.

[45] T. D. Noia, E. D. Sciascio, F. M. Donini, M. Mongiello, A system for principled matchmaking in an electronic marketplace, in: WWW '03: Proceedings of the Twelfth International Conference on World Wide Web, Budapest, Hungary, ACM Press, New York, NY, USA, 2003, pp. 321–330.

[46] OASIS ebXML Joint Committee, Organization for the Advancement of Structured Information Standards, Enabling a global electronic market: ebXML, http://www.ebxml.org.

[47] OASIS Web Services Business Process Execution Language (WSBPEL) Technical Committee, Web Services Business Process Execution Language (WS-BPEL), Oasis Standard, Version 2.0 (April 2007). <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html> (accessed 10.03.2008).

[48] D. Oberle, Semantic Management of Middleware, Ph.D. thesis, Department of Economics and Business Engineering, University of Karlsruhe (TH) (2005).

[49] D. Oberle, S. Lamparter, S. Grimm, D. Vrandecic, S. Staab, A. Gangemi, Towards ontologies for formalizing modularization and communication in large software systems, Journal of Applied Ontology 2 (2) (2006) 163–202.

[50] N. Oldham, K. Verma, A. Sheth, F. Hakimpour, Semantic WS-Agreement partner selection, in: L. Carr, D. D. Roure, A. Iyengar, C. A. Goble, M. Dahlin (eds.), WWW '06: Proceedings of the 15th international conference on World Wide Web, Edinburgh, Scotland, ACM Press, New York, NY, USA, 2006, pp. 697–706.

[51] M. Paolucci, T. Kawamura, T. R. Payne, K. P. Sycara, Semantic matching of web services capabilities, in: I. Horrocks, J. A. Hendler (eds.), Proceedings of the First International Semantic Web Conference (ISWC), Sardinia, Italy, vol. 2342 of Lecture Notes in Computer Science, Springer, 2002, pp. 333–347.

[52] C. Pautasso, G. Alonso, Flexible binding for reusable composition of web services, in: T. Gschwind, U. A. mann, O. Nierstrasz (eds.), Proc. of the 4th Workshop on Software Composition (SC 2005), Edinburgh, Scotland, vol. 3628 of Lecture Notes in Computer Science, Springer, 2005, pp. 151–166.

[53] S. Russel, P. Norvig, Artificial Intelligence - A Modern Approach, 2nd ed., Prentice Hall Series in Artificiall Intelligence, 2003.

[54] R. Sakellariou, V. Yarmolenko, On the flexibility of WS-agreement for job submission, in: MGC '05: Proceedings of the 3rd International Workshop on Middleware for Grid Computing, Grenoble, France, ACM Press, New York, NY, USA, 2005, pp. 1–6.

[55] B. Schnizler, D. Neumann, D. Veit, C. Weinhardt, Trading grid services - a multi-attribute combinatorial approach, European Journal of Operational Research 187 (3) (2008) 943–961.

[56] A. P. Sheth, C. Ramakrishnan, C. Thomas, Semantics for the semantic web: the implicit, the formal and the powerful, International Journal on Semantic Web and Information Systems 1 (1) (2005) 1–18.

[57] T. Skylogiannis, G. Antoniou, N. Bassiliades, G. Governatori, DR-NEGOTIATE - a system for automated agent negotiation with defeasible logic-based strategies, in: EEE '05: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05), Washington, DC, USA, IEEE Computer Society, Los Alamitos, CA, USA, 2005, pp. 44–49.

[58] D. Trastour, C. Bartolini, C. Preist, Semantic web support for the business-to-business e-commerce pre-contractual lifecycle, Computer Networks 42 (5) (2003) 661–673.

[59] United Nations Development Programme, United nations standard products and services code, series 10 (2007). <http://www.unspsc.org> (accessed 14.03.2008).

[60] United Nations Economic Commission for Europe, UN/EDIFACT Standard. <http://www.unece.org/trade/untdid/welcome.htm> (accessed 10.03.2008).

[61] A. Uszok, J. M. Bradshaw, R. Jeffers, KAoS: A Policy and Domain Services Framework for Grid Computing and Semantic Web Services, in: Proceedings of the Second International iTrust Conference, Oxford, UK, vol. 2995 of Lecture Notes in Computer Science, Springer, 2004, pp. 16–26.

[62] M. P. Wellman, J. Doyle, Modular utility representation for decision-theoretic planning, in: J. Handler (ed.), Proceedings of 1st International Conference on AI Planning Systems (AIPS-92), College Park, Maryland, USA, Morgan Kaufmann, 1992, pp. 236–242.

[63] World Wide Web Consortium (W3C), W3C Semantic Web Activity. <http://www.w3.org/2001/sw/> (accessed 17.03.2008).

[64] World Wide Web Consortium (W3C), Web ontology language (OWL), W3C Recommendation (February 2004). <http://www.w3.org/2004/OWL/> (accessed 10.03.2008).

[65] World Wide Web Consortium (W3C), Extensible Markup Language (XML) 1.1, 2nd edition, W3C Recommendation (August 2006). <http://www.w3.org/TR/xml11> (accessed 10.03.2008).

[66] World Wide Web Consortium (W3C), Web services policy framework 1.5, W3C Recommendation (September 2007). <http://www.w3.org/TR/ws-policy> (accessed 10.03.2008).

[67] World Wide Web Consortium (W3C), SPARQL Query Language for RDF, W3C Recommendation (January 2008). <http://www.w3.org/TR/rdf-sparql-query/> (accessed 17.03.2008).

[68] L. A. Zadeh, Fuzzy sets, Information and Control 8 (1965) 338–353.