A Policy Framework for Trading Configurable Goods and Services in Open Electronic Markets

Steffen Lamparter, Anupriya Ankolekar, Rudi Studer
Institute AIFB
University of Karlsruhe (TH)
Karlsruhe, Germany

Daniel Oberle
SAP Research
CEC Karlsruhe
Germany

Christof Weinhardt
Institute of Information Systems and Management
University of Karlsruhe (TH)
Karlsruhe, Germany

ABSTRACT

In recent years, electronic markets have gained much attention as institutions to allocate goods and services efficiently between buyers and sellers. However, calculating suitable allocations between buyers and sellers in such markets can easily become very tricky, particularly if the services and goods involved are complex and described by multiple attributes. Since such trading objects typically provide various configurations with different corresponding prices, complex pricing functions and purchase preferences have to be taken into account when computing allocations. In this paper, we present a policy description framework that draws from utility theory to capture configurable products with multiple attributes. The framework thus allows the declarative description of seller pricing policies as well as buyer preferences over these configurations. As part of the framework, we present a machine-processible representation language for the policies and a method to evaluate different trading object configurations based on these policies. In addition, we show how policies can be aggregated and how the framework can be applied in a Web service selection scenario.

Categories and Subject Descriptors

K.4.4 [**Computers and Society**]: Electronic Commerce; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages; H.3.5 [**Information Storage and Retrieval**]: Online Information Services

General Terms

Algorithms, Theory, Languages

Keywords

Policy, Utility Theory, Ontology

ICEC'06, August 14-16, 2006, Fredericton, Canada.

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. The design of market platforms mainly involves two components [27]: a *communication language* which defines how bids (i.e. offers and requests) can be formalized and submitted to the market mechanism, and an *outcome determination* by means of an allocation (i.e. who gets which service), a pricing schema and a payment component. The design of the communication language is a non-trivial task since it requires mutual understanding between different participants in the market and it involves trading object which are offered by multiple parties (sellers) with different attributes and under different conditions. This is particularly true for configurable goods and services, such as computers and Web-based services. 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. 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).

In this paper, we consider the problem of designing a policy framework that enables the expression of such pricing as well as scoring policies. The framework has to meet several requirements (emphasized): First, since we are dealing with various configurable products, the policy framework must be able to describe ***multi-attribute requests and offers***, i.e. requests and offers that involve multiple attributes beyond just the price, such as quality criteria. Second, pricing as well as scoring policies require the ***expression of functions*** that map configurations to a pricing or a preference structure, respectively. Note that 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.

Third, since pricing policies have to be communicated to the buyer and/or scoring policies to the seller (e.g. in a procurement auction), ***standards-based interoperability*** becomes a crucial issue. Standardized syntax and semantics is particularly essential in open markets, where participants may use highly heterogenous information formats, where buyers and sellers dynamically join or leave the market, and where products and services are highly differentiated and change frequently. For meeting the interoperability requirement our technology of choice are ontologies. Ontologies are also powerful enough to meet the requirements "multi-attribute requests and offers" and "expression of functions" because of the underlying logic and rule mechanism. This prevents us from using additional languages and technologies and simplifies our framework without loss of functionality or expressivity.

In the following, we introduce a policy framework that meets the requirements above. The key contribution of this work is to show how quantitative preferences can be modeled within a declarative framework to include them in the reasoning process. The approach should be as expressive as possible while adhering to internet standards. Before presenting the framework we review related work to determine the extent to which the requirements are already supported by existing work (Section 2). Since our approach is based on *Utility Function policies* we briefly sketch the fundamentals of utility-based policy representation in Section 3. Subsequently, in section 4, we present the ontology formalisms and ontology framework, before discussing how such policies can be declaratively represented by means of ontologies and how policies can be enforced using a semantic framework in Section 5. In Section 6 we show how policies expressed in our framework can be aggregated, before we come up with a concrete application of the approach in Section 7. The implementation introduces an architecture for the dynamic selection of Web services based on our policy framework. Section 8 concludes the paper with a brief outlook.

2. RELATED WORK

In this section, we present various existing approaches in electronic markets for modeling buyer preferences and seller offerings and discuss the extent to which they meet the requirements specified in Section 1. Table 1 summarizes the approaches discussed in terms of which of the four requirements they support (indicated by check marks).

One of the first attempts to exchange order information within electronic markets was the Electronic Data Interchange (EDI) protocol, which serializes 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 not addressing the interoperability requirement.

More recent approaches, such as WS-Policy[10], EPAL [9] and WSPL[8], use XML (eXtensible Markup Language) [35] 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. WS-Agreement [1] is another XML-based specification that can be used to express different valuations for configurations, however, only with discrete attributes. An approach to extend WS-Agreement for expressing continuous functions is presented in [31]. However, the XML-annotations still lack formal semantics.

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. This is the approach followed by KAoS [34] and REI [20], which are policy languages that allow the definition of multi-attributive policies with constraints on attributes. However, these approaches are limited in that they always evaluate either to true or false and thus cannot express the scoring or pricing functions required for configurable products. In this context, more expressivity is provided by SweetDeal [14] and DR-NEGOTIATE [33]. 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 semantic of the syntax is not yet standardized which obstructs 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).

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 (cf. [28], [5]). However, they assume a closed environment and therefore, even if they do use XML-based bidding languages [3], they do not deal with interoperability issues. Many B2B scenarios use standardized product and service taxonomies, such as UN/SPSC[1], CPV[2] or the MIT Process Handbook [24]. However, these taxonomies are static and require the introduction of a new subclass in the hierarchy for each new product configuration. They are therefore clearly inapplicable in our context.

Our approach draws from utility theory to express scoring and pricing functions of market participants within an ontology-based policy framework. Our policy framework is based on existing Internet standards, namely XML for serializing request and offer documents, OWL (Web Ontol-

[1]United Nations Standard Products and Services Code (`http://www.unspsc.org`)

[2]Common Procurement Vocabulary (`http://simap.eu.int/nomen/nomenclature\_standards\_en.html`)

Approach	Requirement		
	Multi-attributive	Functions	Standards-based Interoperability
EDI/EDIFACT	(✓)	(✓)	-
XML-based Languages	✓	-	(✓)
KAoS/REI	✓	-	✓
SweetDeal/DR-NEGOTIATE	✓	(✓)	(✓)
Product/Service Catalogs	-	-	✓
Bidding Languages for CA	✓	✓	-
CPML	✓	✓	(✓)
Our Approach	✓	✓	✓

Table 1: Languages for specifying offers and requests.

ogy Language) [36] and DL-safe SWRL rules (Semantic Web Rule Language) [16, 26] to formalize ontology axioms. Thus, the exchanged documents can be interpreted by standard inference engines. Furthermore, to facilitate integration between offer/request specifications that use heterogenous ontology concepts, we base our policy ontology on the foundational ontology DOLCE [25]. DOLCE provides a high degree of axiomatisation (exact definition) of the policy ontology terms, advantages which carry over to the policy ontology we present.

3. UTILITY-BASED POLICY REPRESENTATION

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. Kephart et. al. [30, 22] 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 [21], 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 [22].

In the context of electronic markets, Utility Function policies can be used on the 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 Utility Function policies enable the compact representation of the pricing or cost function on the seller-side, pricing information can be communicated to the customers in a very efficient way, i.e. with only one message.

For our policy language we use the following simple utility model. Assume alternatives (e.g. configurable trading objects) are described by a set of attributes $X = \{X_1 \dots X_n\}$. Attribute values x_j of an attribute X_j are either discrete, $x_j \in \{x_{j1}, \dots, x_{jm}\}$, or continuous, $x_j \in [min_j, max_j]$. Then the cartesian product $\mathcal{O} = X_1 \times \dots \times X_n$ defines the potential configuration space, where $o \in \mathcal{O}$ 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 $o_1 \in \mathcal{O}$ is preferred to $o_2 \in \mathcal{O}$ if $o_1 \succeq o_2$. The preference structure can be derived from the value function $V(o)$, where the following condition holds: $\forall o_a, o_b \in \mathcal{O} : o_a \succeq o_b \Leftrightarrow V(o_a) \geq V(o_b)$. In order to calculate the valuations $V(o)$ we apply an additive utility model, where the value functions defined in equation 1 below are applied to aggregate the valuations derived from the individual attributes $X_1, \dots, X_n$. The weighting factor λ_j is normalized in the range $[0, 1]$ and represents the relative importance of an attribute j.

$$V(x) = \sum_{j=1}^{n} \lambda_j v_j(x_j), with \sum_{j=1}^{n} \lambda_j = 1 \quad (1)$$

For the additive value function above, we assume mutual preferential independence between the attributes [21]. Under this assumption, we can easily aggregate the utility functions $v_j(x_j)$ of the individual attributes j to obtain the overall valuation of a configuration. However, in real markets often the preferential independency does not hold. In order to capture this dependency the dependent attributes $X_k, \dots, X_l \in X$ are treated as one single attribute j^* in our model, where the utility function is modeled as a complex (higher dimensional) function $v_{j^*}(x_k, \dots, x_l)$. Note that while this approach allows modeling of dependent attributes in a simple way using our policy language, the specification of the function v_{j^*} might be complicated. However, we assume extensive methodology and tool support in the preference elicitation process (cf. [7]).

In the next sections, we show how such an utility-based approach can be declaratively modeled within a system, where both available trading objects as well as policies are stored in a knowledge base and then queries are issued to derive relevant information (e.g. prices, product rankings, etc.).

4. A POLICY DESCRIPTION FRAMEWORK FOR ELECTRONIC MARKETS

For declaratively modelling our utility-based approach we apply ontologies as the state of the art for meeting requirement 3 (standard-based interoperability). Before introducing our ontology for representing Utility Function policies in section 5, we first discuss the underlying formalisms as well as the upper-level modules the ontology is based on. Therefore, section 4.1 presents the ontology as well as rule language and section 4.2 the general ontology framework which is based on the foundational ontology DOLCE.

Module	Concept label	Usage
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 (in this work implemented as datatypes)
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 affaires 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

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.

4.1 Ontology formalism

In recent years, *ontologies* became an important technology for knowledge sharing in distributed, heterogeneous environments, particularly in the context of the *Semantic Web*[3]. An ontology is a set of logical axioms that formally define a shared vocabulary [15]. By committing to a common ontology, sotware agents can make assertions or ask queries that are understood by the other agents.

In order to guarantee that these formal definitions are understood by other parties (e.g. in the web), the underlying logic has to be standardized. The Web Ontology Language (OWL) standardized by the World Wide Web Consortium (W3C) is a first effort in this direction [36]. OWL-DL is a decidable fragment of OWL and is based on a family of knowledge representation formalisms called *Description Logics (DL)* [2]. Consequently, our notion of an ontology is a DL knowledge base expressed via RDF/XML syntax to ensure compatibility with existing World Wide Web languages. The meaning of the modeling constructs provided by OWL-DL like concepts, relations, datatypes, individuals and data values is formally defined via a model theoretic semantics, 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 [18]. Thereby, the meaning of an axiom defines certain constraints on the model. For example, we can define that the concept *Book* is a subconcept of *Product* (i.e. $Book \sqsubseteq Product$). In this case, the interpretation of *Book* has to be a subset of the interpretation of *Product*, i.e. the set of objects that are books is a subset of the set of objects that are products ($Book^{\mathcal{I}} \subseteq Product^{\mathcal{I}}$).

In order to define our policy ontology, we require additional modeling primitives not provided by OWL-DL. For example, we have to model triangle relations between concepts, such as the *isAssignedTo* relation that says an *Attribute Value* is assigned to a *Valuation* in case there is a *Function* with *Attribute Value* as domain and *Valuation* as range. In contrast to OWL, rule languages can be used to express such triangle relation. The Semantic Web Rule Language (SWRL) [16, 17] allows us to combine rule approaches with OWL. Since reasoning with knowledge bases that contain arbitrary SWRL expression usually becomes undecidable [16], we restrict ourself to *DL-safe* rules [26]. DL-safe rules keep the reasoning decidable by placing constraints on the format of the rule, namely each variable occurring in the rule must also occur in a non-DL-atom in the body of the rule. This means the identity of all objects referred to in the rule has to be known explicitly. Thus, the rule in the example above can be formalized as follows:

$$isAssignedTo(x, y) \leftarrow AttributeValue(x), Valuation(y), Function(z), domain(z, x), range(z, y)$$

Obviously, this rule is not DL-safe, since x, y, and z occur only in DL-atoms. However, the rule can be made DL-safe by adding the non-DL-atoms $\mathcal{O}(x), \mathcal{O}(y)$, and $\mathcal{O}(z)$ to the body, which ensure that the variables refer only to *known* objects, i.e. individuals in the knowledge base. Since we deal only with known instances in our application, we do not explicitly mention the non-DL-atoms $\mathcal{O}(x)$ in the following. To query and reason over a knowledge base containing OWL-DL as well as DL-safe SWRL axioms we use the KAON2 inference engine[4].

For the reader's convenience we define DL axioms informally via UML class diagrams[5], where UML classes correspond to OWL concepts, UML associations to object properties, UML inheritance to subconcept-relations and UML attributes to OWL datatype properties [6]. For representing rules we rely on the standard rule syntax as done in [17, 26]. In the following, SWRL rules are labelled by $R1, \ldots, Rn$.

4.2 Modeling Basis

Our policy description framework consists of several ontology modules. These modules are arranged in three layers: (i) As a modeling basis, we rely on the domain-independent upper-level foundational ontology DOLCE [25]. By capturing typical *ontology design patterns* (e.g. location in space and time), foundational ontologies provide basic concepts

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

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

[5] The entire ontology is also available in RDF/XML serialization at `http://ontoware.org/projects/emo`

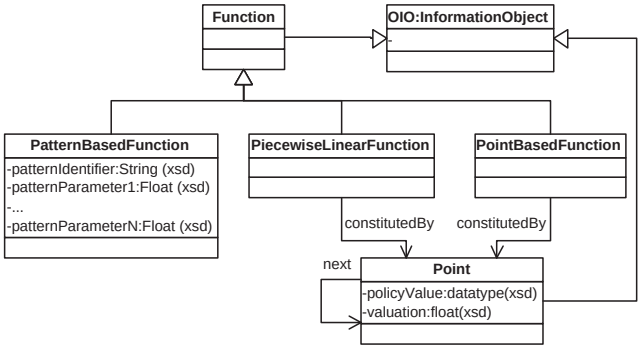

Figure 1: Modeling value functions

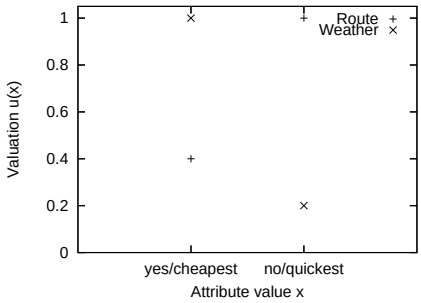

Figure 2: Example of a point-based value function

and associations for the structuring and formalization of application ontologies. 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. Because of space restrictions we omit a detailed description of DOLCE. The DOLCE concepts that are directly used for alignment of our ontology are briefly introduced in Table 2. A detailed description of DOLCE and its modules is given in [25, 12]. (ii) As a second layer we introduce the *Core Policy Ontology* that extends the upper-level ontology modules by introducing concepts and associations that are fundamental for formalizing policies. (iii) 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 the next section, we focus on the Core Policy Ontology and show how policies and configurations are modeled based on DOLCE and the logical formalism introduced above.

5. CORE POLICY ONTOLOGY

In the following, an ontology for modeling policies is introduced that meets the requirements discussed in section 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. Secondly, based on these functions, we show how the DOLCE ontology module Description & Situation is applied to model product configurations and policies. Finally, we introduce interpretation rules that evaluate the configurations according to the specified policies.

5.1 Valuation Functions

As discussed in Section 3, preferences as well as pricing information are expressed via functions that map configurations 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 [23]. 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*[6] is a specialization of *OIO:InformationObject* which represents abstract information that exists in time and is realized by some entity [13]. Currently our framework supports three ways of defining functions: **(i)** *Functions* can be modeled by specifying sets of points in $\mathbb{R}^2$ 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, like in the case of *Response Time*. **(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 contained in the SWRL specification can be used. The different ways of declarative modeling functions are discussed next in more detail.

5.1.1 Point-based Functions

As depicted in Figure 1, *PointBasedFunctions* are *Functions* that are *constitutedBy* a set of *Points*. Thus, the property *policyValue* refers to exactly one attribute value and the property *valuation* to exactly one utility measure that is assigned to this attribute value.[7] Both properties are modeled by OWL datatypes. OWL datatypes mainly rely on the non-list XML Schema datatypes [4]. Depending on the attribute, *policyValue* either points to a *xsd:string*, *xsd:integer* or *xsd:float*. A *valuation* is represented by a *xsd:float* between 0 and 1 or by $-\infty$.

In our route planing example introduced in Section 1, a requester might specify her preferences w.r.t. the service property *Weather* 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 route calculation can be defined with *Points* ("*quickest*", 1) and cheapest ("*cheapest*", 0.4). These mappings are illustrated in Figure 2.

5.1.2 Piecewise Linear Functions

In order to support definition of *Functions* on continuous properties too, we introduce *PiecewiseLinearFunctions* as

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

[7]Note that in case we have dependent attributes and thus complex value functions $v_{j^*}(x_k, \ldots, x_l)$ (cf. Section 3) we require more than one datatype property, i.e. $policyValue_k, \ldots, policyValue_l$.

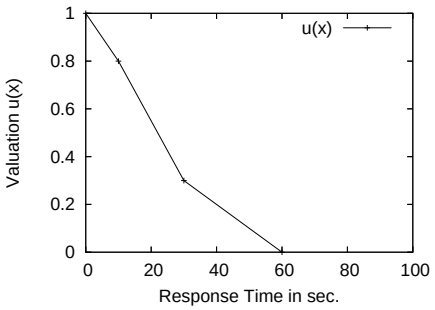

Figure 3: Example of a piecewise linear value function

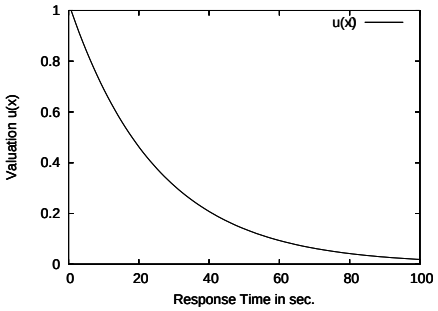

Figure 4: Example of a pattern-based valuation function

shown in Figure 1. To express *PiecewiseLinearFunctions*, we extend the previous approach by the relation *next* between two *Points* with adjacent x-coordinates.

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

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

This equation is formalized using rule R1. To achieve this we exploit the math as well as the comparison built-in predicates provided by SWRL[8].

$$\begin{aligned} &cal_v(v, x, x_1, y_1, x_2, y_2) \leftarrow \\ &\quad subtract(t_1, y_1, y_2), subtract(t_2, x_1, x_2), divide(t_3, t_1, t_2), \\ &\quad subtract(t_4, x, x_1), multiply(t_5, t_3, t_4), add(v, t_5, y_1), \\ &\quad lessOrEqualThan(x_1, x), lessThan(x, x_2) \end{aligned} \quad \text{(R1)}$$

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.

5.1.3 Pattern-based Functions

Alternatively, value functions 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* is a *xsd:string* that uniquely refers to a specific SWRL predicate. A *patternParameter* is a *xsd:float* that 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. Since universal quantification in rule bodies is not expressible in SWRL, 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 R2), each *pattern* is identified by a hard-coded internal string. This is required to specify in the ontology, which pattern is assigned to a certain attribute. Thus, in order to find out which *pattern*-predicate is applicable, the *patternIdentifer* specified in the policy is handed over to the *pattern*-predicate by using the first argument and then it is compared to the internal identifier. If the two strings are identical the predicate is applied to calculate the corresponding *valuation* of a certain *policyValue*.

As an example, we again focus on the attribute response time 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). Axiom R2 formalizes the pattern. The internal identifier in this example is *'id:exp'*. The corresponding comparison is done by the SWRL built-in *equal*, which is satisfied iff the first argument is the same as the second argument.

$$\begin{aligned} &pattern(id, x, p_1, \ldots, p_n, v) \leftarrow \\ &\quad String(id), PolicyValue(x), Valuation(v), \\ &\quad equal(id, "id:exp"), multiply(t_1, p_2, x), \\ &\quad pow(t_2, "2.70481", t_1), multiply(v, p_1, t_2) \end{aligned} \quad \text{(R2)}$$

SWRL supports a wide range of mathematical built-in predicates (cf. [17]) 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.

5.2 Modeling Policies and Configurations

As discussed in Section 3, we formalize preferences of a user as well as pricing information of a provider by means of

[8]For the sake of readability, we use a predicate with arity five. Techniques for reifying higher arity predicates are well known [19].

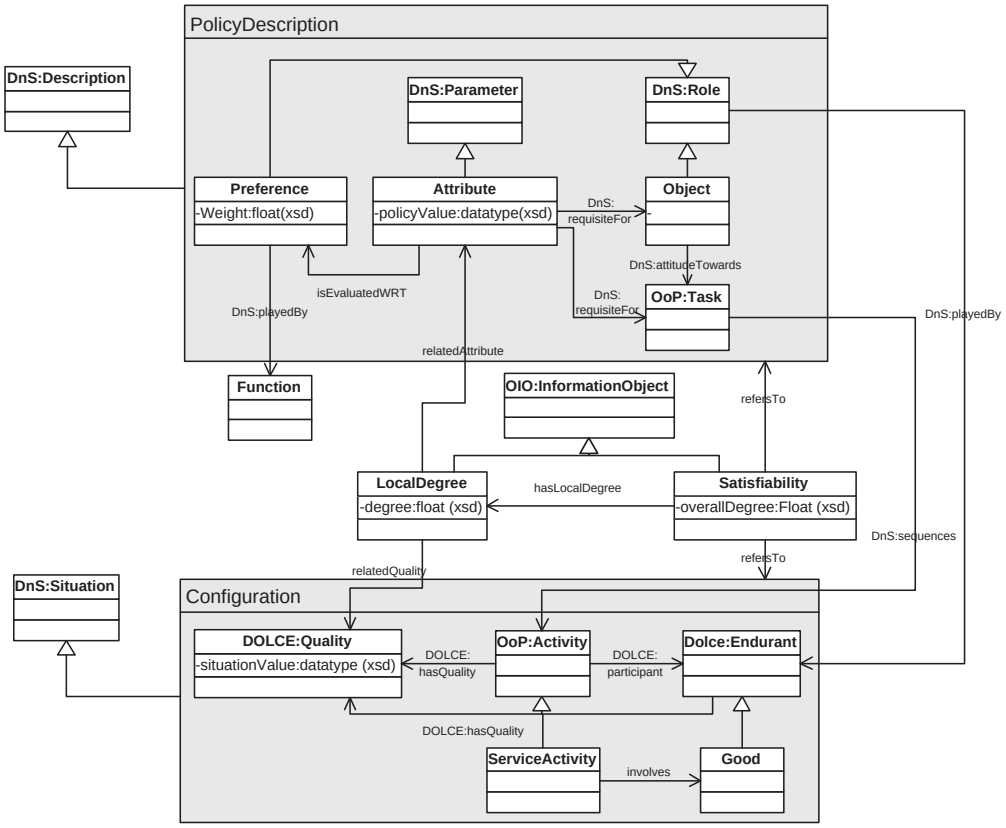

Figure 5: Policy description framework. To improve the readability we illustrate certain relations by plotting UML classes within other UML classes: The class *PolicyDescription* has a *DnS:defines*-relation and the class *Configuration* a *DnS:settingFor*-relation to each contained class.

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 [11]. Such a theory is required to reflect the fact that a certain configuration can be considered as more or less desirable depending on the scoring policies of a buyer or that a certain 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* composing a *DnS:Description*, i.e. the context in which *Situations* are interpreted. As depicted in Figure 5, we specialize the *DnS:Description* to a *PolicyDescription* that can be used to evaluate concrete *Configurations* which are modeled as *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.

5.2.1 Configuration

In a first step, we define the ground entities that describe a *DnS:Situation*. In our context, such *DnS:Situations* reflect configurations of concrete goods or services. Hence, we model *Configuration* as a subclass of *DnS:Situation* as shown in Figure 5. Since there are different ways of defining goods and services, a generic approach is used in this work, where a concrete *Good* is represented by an instance of *DOLCE:Endurant* and a service by a combination of *DOLCE:Endurants* and *OoP:Activities* as done in [11]. Specializations of *OoP:Activities* capture *ServiceActivities* like *RoutePlanning*. Specializations of *DOLCE:Endurants* represent the objects *involved* in such a *ServiceActivity* (e.g. inputs and outputs). Moreover, *DOLCE:Endurants* as well as *OoP:Activities* have *DOLCE:Qualities* with a datatype property *situationValue*.

This means a concrete route planing service is represented by an instance of the following ontology: we specialize *ServiceActivity* to *RoutePlanningActivity* with the *DOLCE:Qualities WeatherInformationQuality*, *ResponseTimeQuality* and *AvailabilityQuality*. In addition, the *RoutePlanningActivity involves* a *ServiceOutput* which specializes *Good*. *ServiceOutput* is associated to a *RouteQuality* that defines whether the output is the cheapest or the quickest route. Note that a *DOLCE:Quality* in a concrete configuration has exactly one *situationValue*-property.

5.2.2 Policy Description

In a second step, the descriptive entities are specialized in order to define policies that can be used to specify scoring functions of the buyer as well as pricing functions of configurable trading objects. As depicted in Figure 5, policies are modeled as specialization of *DnS:Description*, called *PolicyDescription*, which *DnS:defines* a *DnS:Role* representing the *Object* on which the policy is defined, e.g. this could be a certain type of good or the output of a service. Since they are modeled as *DnS:Roles*, policies can be defined on an abstract level without referring to a concrete *DOLCE:Endurant*. For instance, preferences can be defined for a certain product category (*DnS:Object*) such as computers in general and then all *DOLCE:Endurants* that play the role of computers in a certain *Situation* are evaluated according to this preferences. Furthermore, a *PolicyDescription* could also regulate a *OoP:Task*. This is for example the case when talking about web services. A route planing service might execute a *RoutePlanningTask* where the *Input* (specialization of *Object*) is a certain destination and the *Output* (specialization of *Object*) is the calculated route.

However, as discussed in section 3, the configurations are preferred to varying degrees depending on the concrete properties of the trading object. We model this by introducing the *DnS:Parameter Attribute*, which is a *DnS:requisiteFor* a *Object* or *OoP:Task*. *Attributes* have a datatype property *policyValue* pointing to all possible attribute values. Further, each *Attribute* is assigned to a certain preference structure. As discussed above, preference structures on attributes are imposed by *Functions*. *Functions* are *OIO:InformationObjects* (cf. Figure 1) that play the role of *Preferences* in a *PolicyDescription* and define how *policyValues* are mapped to *valuations*. In other words, a policy defines which *Function* should be used in which context (i.e. for which attribute). Besides *Functions*, *Preferences* also define the relative importance of the given *Attribute*.

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

5.2.3 Policy Evaluation

With our approach, policies that contain *Functions* 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 traditional *DnS:satisfies*-relation between a *DnS:Situation* and *DnS:Description* stemming from DnS is not sufficient any more since additional information about the degree of conformity has to be captured. Ontologically, this requires putting in relation the *PolicyDescription*, a concrete *Configuration* and an *overallDegree* that represents the valuation to which the latter satisfies the former. As tertiary relations cannot be modeled with the formalism at hand directly, the *OIO:InformationObject Satisfiability* is introduced to link the three entities. We use the relation *refersTo* to identify the *PolicyDescription* as well as the *Configuration* for which the datatype property *overallDegree* represents the valuation. Figure 5 sketches this modeling approach.

In line with the utility model defined in equation 1, we first calculate the valuation for each attribute individually and then aggregate the individual valuations to the *overallDegree*. For representing individual valuations for attributes we introduce the concept *LocalDegree* which is also an *OIO:InformationObject* in terms of DOLCE. *LocalDegree* links each *DOLCE:Quality* of the *Configuration* to a certain *Attribute* in the *PolicyDescription* by using the associations *relatedAttribute* as well as *relatedQuality*.

On this basis, the property *degree*, which can be interpreted as the valuation a single attribute contributes to the overall valuation, is calculated as follows: depending on which *Function DnS:plays* the role of *Preference* for a certain *Attribute*, one of the rules below is used. In order to abbreviate the following rule definitions, we first define the shortcut relation *isDeterminedBy* between a *LocalDegree* and the *Function* that specifies the *Preferences* for the *Attribute* related to the *LocalDegree*.

$$\begin{aligned} &isDeterminedBy(x, f) \leftarrow \\ &\quad relatedAttribute(x, a), isEvaluatedWRT(a, p), \\ &\quad DnS{:}playedBy(p, f) \end{aligned} \tag{R3}$$

In case of *PointBasedFunctions* we look up the *situationValue* in the *Configuration* and compare this value with the *policyValues* of all *Points* defined by the *Function*. If the *policyValue* of a *Point* p matches the *situationValue*, the property *valuation* of p determines the *degree*. For instance, if the *situationValue* for *RouteQuality* is determined by the *xsd:string* "quickest" and the *PointBasedFunction* is given by ("*quickest*", 0.6) and ("*cheapest*", 0.4) we derive a *degree* of 0.6 since only the *policyValue* "quickest" matches the *situationValue*. This is formalized by Rule R4.

$$\begin{aligned} &degree(ld, v) \leftarrow \\ &\quad isDeterminedBy(ld, f), \mathbf{PointBasedFunction}(f), \\ &\quad constitutedBy(f, po), policyValue(po, pv), \\ &\quad relatedQuality(ld, q), situationValue(q, y), equals(pv, y), \\ &\quad valuation(po, v) \end{aligned} \tag{R4}$$

Correspondingly, the Rules R5 and R6 can be used to evaluate *PiecewiseLinearFunctions* and *PatternBasedFunctions*, respectively. Rule R5 uses the cal_v-predicate (defined in Rule R1) and Rule R5 the *pattern*-predicate (defined in Rule R2) to determine allowed mappings between *policyValues* and *valuations*.

$$\begin{aligned} &degree(ld, v) \leftarrow \\ &\quad isDeterminedBy(ld, f), \mathbf{PiecewiseLinearFunction}(f), \\ &\quad \bigwedge_{i \in \{1,2\}} (constitutedBy(f, po_i), policyValue(po_i, pv_i), \\ &\quad valuation(po_i, v_i)), next(po_1, po_2), relatedQuality(ld, q), \\ &\quad situationValue(q, y), cal_v(v, y, pv_1, v_1, pv_2, v_2) \end{aligned} \tag{R5}$$

$$\begin{aligned} &degree(ld, v) \leftarrow isDeterminedBy(ld, f), \\ &\quad \mathbf{PatternBasedFunction}(f), patternIdentifier(f, id), \\ &\quad relatedQuality(ld, q), situationValue(q, y), \\ &\quad \bigwedge_{i \in \{1,..,n\}} (patternParameter_i(f, p_i)), \\ &\quad pattern(id, y, p_1, \ldots, p_n, v) \end{aligned} \tag{R6}$$

The valuation derived from the individual attributes is weighted according to their relative importance defined by the concept *Weight*. The weights λ_i of the individual attributes i have to be normalized between 0 and 1. This is done by means of the formula $\frac{1}{n}\sum_{i=1}^{n}\lambda_i$, which is evaluated within rule R7. Based on the normalized weights, Rule R7 calculates the *overallDegree* by encoding equation 1.

$$\begin{aligned} &overallDegree(s,v) \leftarrow \\ &\quad PolicyDescription(d), refersTo(s,d), Configuration(c), \\ &\quad refersTo(s,c), \bigwedge_{i\in\{1,\ldots,n\}} (hasLocalDegree(s,ld_i), \\ &\quad isDeterminedBy(ld_i,f_i), weight(f_i,w_i)), \\ &\quad sum(g,w_1,\ldots,w_n), \bigwedge_{i\in\{1,\ldots,n\}} (div(r_i,w_i,g), \\ &\quad localDegree(ld_i,v_i), mul(k_i,v_i,r_i)), \\ &\quad sum(v,k_1,\ldots,k_n) \end{aligned} \tag{R7}$$

As an example, we assume the *Configuration* s of a route planing service, which returns the quickest route while considering traffic information. Further, a response time of 20 sec. is guaranteed. Based on the example *Functions* above this leads to local *degrees* of 1 for the *Attribute Weather*, 1 for *Traffic*, 0.47 for *Response Time* and 0.5 for *Route*, respectively. Moreover, we assume that the PolicyDescription contains the weights of $2, 2, 1, 1$ for the individual *Attributes*. Now, we can query the knowledge base containing the configuration c as well as the PolicyDescription d (e.g. by using SPARQL[9]) to get the *overallDegree* of a *Satisfiability* instance that *refersTo* c and d. In the example this would result in a *overallDegree* of 0.83. Alternatively, the following SPARQL-query can be used to derive a ranking of all *Configurations* in the knowledge base according to a certain 'ScoringPolicy_d':

BASE <*http://example.org/*>
PREFIX *poem:*<*http://ontoware.org/projects/emo/0.1/*>
SELECT *?Configuration ?overallDegree*
WHERE {
?Satisfiability rdf:type poem:Satisfiability .
?Satisfiability poem:refersTo "ScoringPolicy_d" .
?Configuration rdf:type poem:Configuration .
?Satisfiability poem:refersTo ?Configuration .
?Satisfiability poem:overallDegree ?overallDegree .}
ORDER BY *DESC*(*?overallDegree*)

6. 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 policy from such diverse policies. Therefore, policies are first evaluated and the results of this evaluation step are then aggregated.

[9] `http://www.w3.org/TR/rdf-sparql-query/`

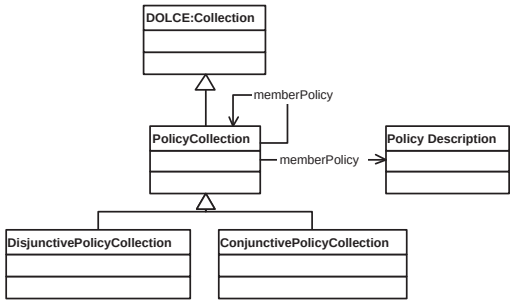

Figure 6: Policy Collection

In traditional policy languages there are two major operators that can be used to combine policies [23, 10]: 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 [37] as follows:

$$\begin{aligned} \top(a,b) &= min(a,b) \text{ for } and\text{-operators} \\ \bot(a,b) &= max(a,b) \text{ for } or\text{-operators} \end{aligned}$$

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 *DnS:Situation* with respect to a set of policies, we adapt the *Satisfiability* concept in a way that it may not only *referTo* a single *PolicyDescription*, but also to a *PolicyCollection*. A *PolicyCollection* is defined as a *DOLCE:Collection* that has exactly two *memberPolicy*-relations pointing either to a *PolicyDescription* or to another *PolicyCollection*. This is formalized using the following DL axioms:

$$\begin{aligned} PolicyCollcection \sqsubseteq\ & DOLCE{:}Collection \sqcap \\ & \exists memberPolicy1.(PolicyDescription \\ & \sqcup PolicyCollection) \sqcap \\ & \exists memberPolicy2.(PolicyDescription \\ & \sqcup PolicyCollection) \end{aligned}$$

$$\begin{aligned} memberPolicy &\sqsubseteq DOLCE{:}member \\ memberPolicy1 &\sqsubseteq memberPolicy \\ memberPolicy2 &\sqsubseteq memberPolicy \end{aligned}$$

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 '$\forall y.memberPolicy(x, y)$' is not expressible in SWRL). However, this is 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 $\top$ defined above.

$$\begin{aligned} &overallDegree(s, d) \leftarrow Satisfiability(s), \\ &\quad refersTo(s, c), ConjunctivePolicyCollection(c), \\ &\quad \bigwedge_{i \in \{1,2\}} (memberPolicyN(p_i), refersTo^{-1}(p_i, s_i), \\ &\quad overallDegree(s_i, d_i)), min(d, d_1, d_2) \end{aligned} \quad \text{(R8)}$$

Note that rule R8 recursively calculates the *overallDegree* of the elements contained in the collection. Rule R8 will only be used if *Satisfiability* refers to a *ConjunctivePolicyCollection*. If it refers to a single *PolicyDescription*, rule R7 will be applied as before.

Analogously, we can define the rule R9 for *DisjunctivePolicyCollections* where the T-conorm $\perp$ is used to calculate the *overallDegree*.

$$\begin{aligned} &overallDegree(s, d) \leftarrow Satisfiability(s), \\ &\quad refersTo(s, c), DisjunctivePolicyCollection(c), \\ &\quad \bigwedge_{i \in \{1,2\}} (memberPolicyN(p_i), refersTo^{-1}(p_i, s_i), \\ &\quad overallDegree(s_i, d_i)), max(d, d_1, d_2) \end{aligned} \quad \text{(R9)}$$

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

7. APPLICATION: DYNAMIC WEB SERVICE SELECTION

In this section, the policy language presented above is used to implement a *service bus* architecture [32, 29]. A service bus can be considered as a component that enables dynamic Web service selection and thus avoids hard-wiring of Web services within a service-oriented implementation of a business process. Dynamic selection of services is a central problem when dealing with service oriented architectures. It 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 7 shows the architecture of our system. 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. 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. Moreover, the company posts a classification rule like R10, which defines a threshold value for acceptable service configurations. In this example, the concept *Acceptable* classifies all service configurations which lead to difference between willingness to pay and price of at least 0.2. This threshold should be realized for all future service invocations.

$$\begin{aligned} &Acceptable(x) \leftarrow Configuration(x), PricingPolicy(pp), \\ &\quad Satisfiability(s2), refersTo(s2, pp), refersTo(s2, x), \\ &\quad ScoringPolicy(sp), Satisfiability(s1), \\ &\quad refersTo(s1, sp), refersTo(s1, x), \\ &\quad overallDegree(s1, score), overallDegree(s2, price), \\ &\quad sub(dif, score, price), greaterEqual(dif, 0.2) \end{aligned} \quad \text{(R10)}$$

   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.

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 those 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. Note that technically, obtaining an offer is a two-step process, where the provider's policy is queried via the WS Metadata Exchange interface and application specific attributes are obtained by calling some *getOffer*-method.

4. Finally, the service bus queries the knowledge base for all instances of *Acceptable* ranked according to the difference between score and price. 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 this case an error message is forwarded to the requester.

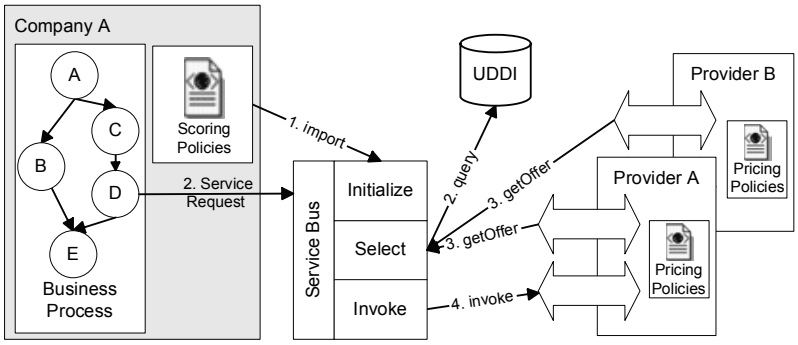

Figure 7: Service Bus Architecture

Note that based on rule R10, the inference engine automatically classifies all configuration instances according to the condition and a simple query can be used to derive all *acceptable* services. Such automatic classification of configurations is particularly important for implementing context-aware decision making. For example, different contexts (current location, time, etc.) may require different scoring policies. In our rule-based framework such *context-rules* can be easily added to the knowledge base.

8. CONCLUSION

In this paper, we provide a formal representation of a utility-based policy framework, which realizes the advantages of *Utility Function policies*, such as preference modeling and inherent conflict resolution, with a purely declarative and standard-based approach. This is essential for flexibility and interoperability of the system within electronic markets. As a use case, it is shown how such a policy framework can be applied for the formal modeling of pricing policies of a provider as well as preferences of a requester. In order to exemplify this approach we present the architecture of a service bus which enables dynamic selection of Web services. We believe that expressing the relations between product/service configurations and prices or the willingness to pay is crucial in electronic markets for configurable trading objects. To the best of our knowledge, there are no formal, declarative and standard-based languages available yet that provide sufficient expressivity to support this.

In a next step, we plan to integrate the policy framework into an larger ontology for expressing offers, requests and agreements in a market. In this context, super- as well as subadditive valuations should be supported through primitives for representing *bundles* and *substitutes*. In addition, we plan to extend the service selection scenario by including negotiation approaches like double auctions. Hence, the ontology has to be mapped to an allocation problem, which efficiently encodes the winner and price determination in the market based on pricing as well as scoring policies.

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 SEKT (IST-506826) and DIP (IST-507483). In addition, we thank our colleague Denny Vrandečić and the reviewers of the ICEC'06 conference for their valuable comments and suggestions.

9. REFERENCES

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

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

[3] M. Bichler and J. Kalagnanam, *Configurable offers and winner determination in multi-attribute auctions*, European Journal of Operational Research **160** (2005), no. 2, 380–394.

[4] Paul V. Biron and Ashok Malhotra, *XML Schema Part 2: Datatypes*, W3C Recommendation. Latest version is available at `http://www.w3.org/TR/xmlschema-2/`, May 2000.

[5] Craig Boutilier and Holger H. Hoos, *Bidding languages for combinatorial auctions*, International Joint Conference on Artificial Intelligence IJCAI'01, 2001, pp. 1211–1217.

[6] Saartje Brockmans, Raphael Volz, Andreas Eberhart, and Peter Löffler, *Visual modeling of OWL DL ontologies using UML*, Proc. of the 3rd International Semantic Web Conference (Hiroshima, Japan) (S.A. McIlraith et al., ed.), Springer LNCS, November 2004, pp. 198–213.

[7] Li Chen and Pearl Pu, *Survey of Preference Elicitation Methods*, Tech. report, Ecole Politechnique Federale de Lausanne (EPFL), 2004.

[8] Anne Anderson et. al., *XACML profile for web services*, `http://xml.coverpages.org/WSPL-draft-xacmlV04-1.pdf`, September 2003.

[9] Paul Ashley et. al., *Enterprise privacy authorization language*, W3C Submission, `http://www.w3.org/Submission/EPAL`, November 2003.

[10] Siddharth Bajaj et. al., *Web services policy framework*, `http://www-128.ibm.com/developerworks/library/specification/ws-polfram`, September 2004.

[11] A. Gangemi, P. Mika, M. Sabou, and D. Oberle, *An ontology of services and serivce descriptions*, Tech. report, Laboratory for Applied Ontology, Rome, Italy, November 2003.

[12] A. Gangemi, M.-T. Sagri, and D. Tiscornia, *A Constructive Framework for Legal Ontologies*, Internal project report, EU 6FP METOKIS Project, Deliverable, 2004, `http://metokis.salzburgresearch.at`.

[13] Aldo Gangemi, Stefano Borgo, Carola Catenacci, and Jos Lehmann, *Task taxonomies for knowledge content*, Metokis deliverable d07, June 2004.

[14] B. Grosof and T. Poon, *Sweetdeal: Representing agent contracts with exceptions using XML rules, ontologies, and process descriptions*, Proceedings of the 12th International Conference on the World Wide Web (WWW 2003) (Budapest, Hungary), May 2003.

[15] T. R. Gruber, *A translation approach to portable ontologies*, Knowledge Acquisition **5** (1993), no. 2, 199–220.

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

[17] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean, *SWRL: A semantic web rule language combining OWL and RuleML*, W3C Submission, avilable at `http://www.w3.org/Submission/SWRL`, May 2004.

[18] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen, *From SHIQ and RDF to OWL: The making of a web ontology language*, Journal of Web Semantics **1** (2003), no. 1, 7–26.

[19] Ian Horrocks, Ulrike Sattler, Sergio Tessaris, and Stephan Tobies, *How to decide query containment under constraints using a description logic*, Proceedings of the 7th International Conference on Logic for Programming and Automated Reasoning (LPAR'2000), 2000.

[20] Lalana 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.

[21] R. L. Keeney and H. Raiffa, *Decisions with multiple objectives: Preferences and value tradeoffs*, J. Wiley, New York, 1976.

[22] J. O. Kephart and W. E. Walsh, *An artificial intelligence perspective on autonomic computing policies*, Proc. of 5th IEEE Int. Workshop on Policies for Distributed Systems and Networks (Yorktown Heights, New York, USA), IEEE Computer Society, 2004, pp. 3–12.

[23] Steffen Lamparter, Andreas Eberhart, and Daniel Oberle, *Approximating service utility from policies and value function patterns*, 6th IEEE Int. Workshop on Policies for Distributed Systems and Networks (Stockholm, Sweden), IEEE Computer Society, June 2005, pp. 159–168.

[24] T. W. Malone, K. Crowston, B. P. Jintae Lee, C. Dellarocas, G. Wyner, J. Quimby, C. S. Osborn, A. Bernstein, G. Herman, M. Klein, and E. O'Donnell, *Tools for inventing organizations: Toward a handbook of organizational processes*, Management Science **45** (1997), no. 3.

[25] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, A. Oltramari, and L. Schneider, *The WonderWeb library of foundational ontologies*, WonderWeb Deliverable D17, Aug 2002.

[26] Boris Motik, Ulrike Sattler, and Rudi Studer, *Query answering for OWL-DL with rules*, Journal of Web Semantics: Science, Services and Agents on the World Wide Web **3** (2005), no. 1, 41–60.

[27] Dirk 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.

[28] Noam Nisan, *Bidding and allocation in combinatorial auctions*, Proceedings of the 2nd ACM conference on Electronic commerce (EC'00) (New York, NY, USA), ACM Press, 2000, pp. 1–12.

[29] Rick Robinson, *Understand enterprise service bus scenarios and solutions in service-oriented architecture - the role of the enterprise service bus*, Tech. report, IBM developerWorks, 2004, `http://www-128.ibm.com/developerworks/webservices/library/ws-esbscen/`.

[30] S. Russel and P. Norvig, *Artificial Intelligence - A Modern Approach*, second ed., Prentice Hall Series in Artificiall Intelligence, 2003.

[31] Rizos Sakellariou and Viktor Yarmolenko, *On the flexibility of WS-agreement for job submission*, Proceedings of the 3rd Intern. Workshop on Middleware for Grid Computing (MGC'05) (New York, NY, USA), ACM Press, 2005, pp. 1–6.

[32] Roy W. Schulte, *Predicts 2003: Enterprise service buses emerge*, Gartner Report (2002).

[33] Thomas Skylogiannis, Grigoris Antoniou, Nick Bassiliades, and Guido Governatori, *DR-NEGOTIATE - A system for automated agent negotiation with defeasible logic-based strategies*, Proce. of the IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05) (Washington, DC, USA), IEEE Computer Society, 2005, pp. 44–49.

[34] Andrzej Uszok, Jeffrey M. Bradshaw, and Renia Jeffers, *KAoS: A Policy and Domain Services Framework for Grid Computing and Semantic Web Services*, Trust Management: Second International Conference, iTrust 2004, Oxford, UK, March 29 - April 1, 2004. Proceedings, LNCS, vol. 2995, Springer, 2004, pp. 16–26.

[35] W3C, *Extensible markup language (XML) 1.1*, `http://www.w3.org/TR/xml11`, February 2004, W3C Recommendation.

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

[37] Lofti A. Zadeh, *Fuzzy sets*, Information and Control **8** (1965), 338–353.