Approximating Service Utility from Policies and Value Function Patterns

Steffen Lamparter, Daniel Oberle
Institute AIFB
University of Karlsruhe, Germany
{lamparter, oberle}@aifb.uni-karlsruhe.de

Andreas Eberhart
Hewlett-Packard
Waldorf, Germany
andreas.eberhart@hp.com

Abstract

Service-oriented computing provides the right means for building flexible systems that allow dynamic configuration and on-the-fly composition. In order to realize this vision, the system must be able to choose the most suitable service from a large and constantly changing number of providers. We present an approach for selecting services based on a coherent conceptual policy model and a service utility measure. Our framework is capable of capturing technical and application specific aspects of services and incorporates them into the decision making process. We assist the user in establishing the utility measure from existing policies by attaching value function patterns to the individual attributes. Drawing from the areas of utility theory, foundational ontology, and electronic markets, our work is a promising approach for unifying the heterogeneous methodologies in the service selection process.

1 Introduction

Service Oriented Architecture (SOA) is an architectural style of developing systems, where services are loosely coupled amongst each other. Loose coupling implies, that it is possible for a requestor to substitute one service with another one or even select a service dynamically. In context of the SOA, policies have recently received a lot of attention and several standards like WS Policy, WS Security, XACML, and others have emerged. The rationale for these languages is that off-the-shelf policy engines enforce policies and use them in the service selection process.

Our work is motivated by the following three observations: Firstly, today's policy languages are driven by very heterogeneous communities, resulting in policy languages with different syntax, semantics, and underlying conceptual models. The multiplicity of description languages in the SOA domain even coined the term WS* which refers to this collection of languages[1]. For instance, the EPAL and XACML specifications greatly overlap and do very similar things in slightly different ways. As a consequence, the user has to learn the different approaches and work with different policy tools. Furthermore, it is not possible to specify a policy that combines privacy and communication security concerns such as: send sensitive content over secured lines only. WS Policy aims at solving this problem by providing a container structure and simple logic framework that allows other policy assertions to be plugged in. Nevertheless, the supported assertions are very simplistic in nature and still require the respective native policy interpreters.

Secondly, it is unclear where policies end and application concerns start. Clearly, security and transactional settings of a service would be specified as policies. Delivery times of a supplier might still be policies, but the cost of an item or even product-specific details such as megapixels measures of digital cameras are definitely considered to be application specific attributes. Nevertheless, when choosing a supplier, all of the attributes mentioned above will play a role in determining a provider and the technical means of invoking its system. For instance, one might choose a more expensive provider over one with a flaky technical infrastructure. However, today's policy frameworks only focus on the low-level technical aspects of a service.

Thirdly, we look at the area of electronic negotiations. In order to negotiate, the value of a trading object has to be derived (e.g. for bidding in an auction). Therefore, more fine-grained preference information than provided by existing policy frameworks is needed. Often multi-attribute utility theory is used to represent preferences and calculate the user's service valuation. In order to define the service utility, theoretically one would have to provide a value for every combination of the service attributes in a n-dimensional space. Clearly, this is not feasible and suitable approximations need to be identified.

Before the three problems outline above are addressed, section 2 introduces the scenario of policy based web service selection in more detail. In section 3 different classes

[1]http://www-106.ibm.com/developerworks/views/webservices/standards.jsp

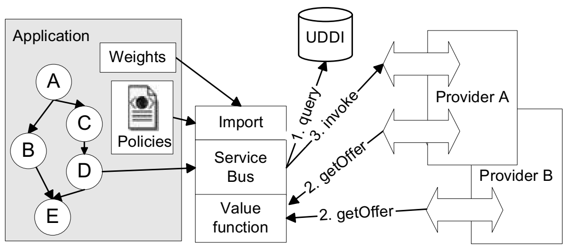


Figure 1. Architecture

of policies are distinguished. In order to express these policy classes by means of a consistent conceptual model, section 4 borrows ideas and methodologies from the area of formal ontology. Basing on this foundation, section 5 introduces a utility model for service offers that allows to express the valuation of application specific and technical attributes. According to Kephart, policies can be viewed as high-level guidelines or directives from human users that influence the behavior of (complex) autonomous systems [13]. Thus, policies implicitly reveal some parts of the human preference structure, which can be transformed into low-level actions in the system. We pick up this idea and show how the value function can be approximated from user policies and weighting information. Thus, policies can be used not only for a pure boolean decision regarding the suitability of a service, but also for estimating the degree of suitability. We provide an example and related work in sections 7 and 8 before concluding.

2 Service Selection

In this section the scenario of dynamic web service selection is introduced. This is a central problem when dealing with service oriented architectures. Figure 1 shows the architecture of the system. On the left side we see the requestor's application, visualized as a workflow. The application also contains information about the weights that represent the relative importance of attributes and policies that define how the application should behave. From within this flow, an external service is requested at step D. Together with the first service request, policy as well as attribute weighting information is sent to the service bus. Policies and weights are converted into a requestor-specific utility function. Note that this step only has to happen once for the initialization of the system. Based on this function the service bus is able to take over the duty of selecting between the potential providers A and B.

Once a request from an application arrives, the service bus first queries a UDDI registry for suitable providers. In the second step, offers from the providers are collected in parallel. In the third step, the service bus selects the optimal service by evaluating the offers according to the utility function and calls the respective provider.

In this paper utility functions are approximated from policies in the context of service selection. However, our approach could be also a useful instrument in many other scenarios.

3 Policy Classes

In this section three major aspects of policy statements are introduced: The deontic modality (section 3.1), the policy constraint (section 3.2), and the scale of the attribute values (section 3.3). This classification can be used to categorize policies and is employed later on in order to draw conclusions about the preference structure that is imposed by a policy. Such a preference structure can be conceived as the set of possible preference relations between different alternatives (e.g. attribute values or services).

3.1 Deontic Modality

Policies are used to control the behavior of a system and therefore, a modal description has to be part of each policy statement. Basically, deontic modalities are normative relations, such as rights, obligations, privileges, non-rights, implicit or explicit permissions, etc.[2] In order to specify guidelines for a system, obligations and recommendations are the most important concepts. Depending on the system architecture, authorizations such as rights or permissions might also be necessary, but they are not the focus of this paper. Our usage of the terms *obligation* and *recommendation* is in line with RFC 2219[3].

Obligation: Obligations define constraints that *must* be met by a system in order to behave properly. E.g. an attribute must have a specific value, otherwise the service does not meet the requirements and cannot be chosen.

Recommendation: Recommendations are policies that do not necessarily have to be met, but the user definitely prefers an offer that complies with a recommendation. That means recommendations can be seen as soft constraints. In contrast to obligations, recommendations also accept attribute values that violate the constraint stated in the policy.

3.2 Policy Constraint

Policies narrow the range of values an attribute may adopt. Therefore, two components are required: *operators* and *constants*. Policy operators are comparative predicates such as $<, >, ==, \neq, \geq, \leq$ that divide the set of possible attribute values into two distinct sets by comparing attribute values with a specific constant: One set containing the allowed values of an attribute j (denoted by $\mathcal{P}_j$) and one set

[2]A detailed discussion about the modalities is given in [9, 17, 8].
[3]http://sunsite.iisc.ernet.in/collection/rfc/rfc2119.html

of attribute values that violate the policy (denoted by $\mathcal{W}_j$). In this context, the set of all attribute values X_j is defined by $X_j = \mathcal{W}_j \cup \mathcal{P}_j$ and the equation $\mathcal{W}_j \cap \mathcal{P}_j = \emptyset$ must hold.

3.3 Scale of Attribute Values

When analyzing policy statements, the scale of the underlying attribute plays an important role. For instance, depending on the type of the scale, some operators cannot be applied in a policy statement or the meaning of operators may change.

In this section we introduce the four main scales used to express the attributes of a service.

Nominal scale: The attribute values are represented by mutually exclusive categories. The categories do not have any mathematical significance. An example for such a scale is the color-attribute of a product which can be valued by *black*, *red*, or *blue*. Because the scale lacks an inherent order, only the policy operators '$==$' and '$\neq$' can be used.

Ordinal scale: The attribute values are assigned to categories which have an inherent order of magnitude, i.e. the order of the categories is defined, but not the exact distance between the categories. The distance between different categories does not have to be uniform. E.g. the quality of a service could be represented by the ordered categories *'very good'*, *'good'* and *'bad'*. While all policy operators can be used, no concrete statement can be made about the distance between the attribute values (at least without any background knowledge).

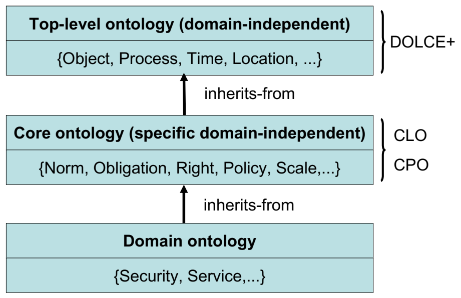
Interval scale: In addition to the properties of the ordinal scale, the interval scale requires equal differences between the categories. However, an interval scale does not have an absolute zero point. A typical example for such a scale is expressing dates where no absolute zero point can be defined. Using this scale a concrete statement about the difference between two categories can be made.

Ratio scale: Using a ratio scale, attribute values are ordered, the distances between the values are equal, and there is an absolute zero point. Examples are attributes like weight, length, etc.

The scale of the constrained attribute determines, together with the deontic modality and the policy constraint, the class of a policy. In the next section we describe a framework that can be used to model the different policy classes using a uniform formalism.

4 A generic policy description framework

We express policies via an ontology. Ontologies formalize concepts and concept relationships (associations) very similar to conceptual database schemata or UML class diagrams [19]. However, ontologies typically feature logic-based representation languages. Those languages come with executable calculi that allow querying and reasoning during run-time. Besides, such formalisms facilitate the conceptual integration of heterogeneous policy efforts by providing well-defined and machine understandable semantics. In this section we present our generic policy description framework that is based on foundational ontologies. Foundational ontologies capture typical *ontology design patterns* (e.g. location in space and time). By providing precise concept definitions they facilitate the conceptual integration of different policy efforts.



Top-level ontology (domain-independent)
{Object, Process, Time, Location, ...}
DOLCE+
inherits-from
Core ontology (specific domain-independent)
{Norm, Obligation, Right, Policy, Scale,...}
CLO
CPO
inherits-from
Domain ontology
{Security, Service,...}

Figure 2. Structure of Policy Description Framework (cf. [8])

As figure 2 shows, the policy description framework used for this work consists of 3 layers: (i) A domain-independent upper-level ontology that provides basic concepts and associations for structuring and formalization of application ontologies. (ii) A *Core Legal Ontology* that represents basic concepts for modeling legal norms. (iii) A *Core Policy Ontology* that extends the core legal ontology by introducing concepts and associations to formalize policies in a well-defined way. The first two components are off-the-shelf ontologies that are used as modeling basis for the construction of domain specific ontologies [6]. Below the three components are described in more detail.

(i) DOLCE+

The foundational ontology DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) provides the basis for the policy description framework used in this paper. Foundational ontologies are high-quality formalizations of domain independent concepts and associations that contain a rich axiomatization of their vocabulary. D&S is an ontology module that extends DOLCE. DOLCE together with D&S is referred to as DOLCE+ and introduces the basic distinction between descriptive and ground entities (as shown in figure 3). A detailed description of DOLCE and D&S can be found in [15] and [7], respectively.

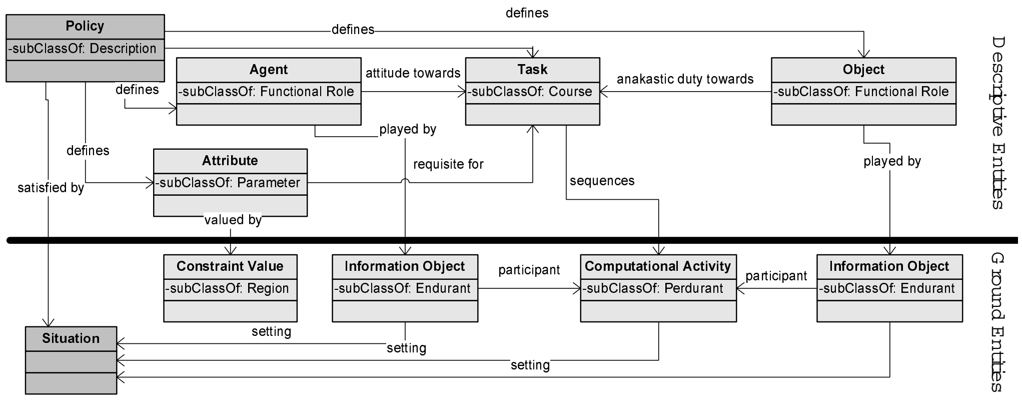


Figure 3. Core Policy Ontology

(ii) Core Legal Ontology.

The Core Legal Ontology (CLO) is a further extension of DOLCE and organizes juridical concepts and relations on the basis of formal properties defined in the DOLCE foundational ontology [8]. The content of a legal regulation, norm, etc. is modelled as a Description[4], while a legal case can be regarded as a concrete Situation in terms of DOLCE+.

Policies and the legal domain deal with similar questions, e.g. the question whether an action is allowed or not. Both, policies as well as legal norms, define general rules which constrain the behavior of complex systems (cf. [5]). Hence, the basic concepts of the Core Legal Ontology meet our requirements of expressing policies and are general enough to harmonize existing policy efforts such as WS-Policy or XACML which all introduce their own vocabulary. Especially important for modeling policies is the fact that Core Legal Ontology supports a wide range of normative relations that can be used to represent the deontic modality of policies.

(iii) Core Policy Ontology.

In order to express policies we have to extend the basic vocabulary with policy specific concepts and relations, while reusing the foundational ontologies as far as possible. This core ontology contains the basic building blocks needed for modeling policies.

Figure 3 sketches the Core Policy Ontology (CPO) in a simplified way. The framework can be used to express the different classes of polices introduced in section 3. Additionally, the application area for each policy has to be defined. The application area states in which situation a policy is relevant and has to be considered.

As shown in figure 3, all concepts of the CPO are subclasses of DOLCE top-level concepts. A policy description consists of the concepts Agent, Task, Object, and Attribute. The entities Agent, Task, and Object allow to define the application area of the policy, while Attribute defines the property of a service that is constrained by the policy. The chosen property is then valued-by a Region (called Constraint Value) that defines the overall range of the attribute values. Regions contain values that are represented by one of the scales described in section 3 (omitted in figure 3). By using the policy constraints (also defined in section 3), the Regions can be specialized in a way that only attribute values are contained, which are valid according to the policy. Thus, the set of valid attribute values $\mathcal{P}_j$ and the set of forbidden attribute values $\mathcal{W}_j$ represent new subordinate regions and $\mathcal{P}_j$ defines the Constraint Value in the policy statement.

Similar to WS-Policy our ontology allows to specify a collection of policies (not shown in figure 3). A collection consists of several policies interpreted together. Our framework allows to combine policies with boolean (and, or) and implication operators (logical and reverse implication). This makes it possible to specify, for instance, that in order to get a positive decision all policies must be valid or alternatively, the validity of one policy from the collection might be enough.

For the identification of the preference structure implied by a policy the class of a policy plays an important role. For policies modelled by means of the framework presented above we can easily determine the class of the policy. This is done by analyzing the relation between Agent and Task, the scale of the Region (e.g. Ordinal Region, Nominal Region, . . .), and the way the Region is constrained (e.g. which operator). By using the Core Policy Ontology we can express a limited number of policy classes in a well-defined and unambiguous way. This makes it a lot easier to draw conclusions regarding the underlying preference structure. Before we show how policy classes and preference structures can be related, we introduce the term preference structure formally and show how such a structure can be modelled by means of utility theory.

[4]Concepts and associations of the ontology are written in sans serif.

5 Utility Model

In order to make appropriate decisions a way to represent user preferences is required. For this purpose we employ a basic utility model similar to those used in [12, 4]. The properties of a service are described by a set of attributes $X = \{X_1 \ldots X_n\}$. This set of attributes should cover all important aspects of the service. Attribute values are either discrete, $x_j \in \{x_{j1}, \ldots, x_{jm}\}$, or continuous, $x_j \in [min_j, max_j]$[5]. Based on the Cartesian Product $\Omega = \{X_1, \times \cdots \times, X_n\}$ of the attributes, the potential outcome space that has to be considered when making a decision can be defined as $\mathcal{O} \subseteq \Omega$. Hence, a service is described by one of the possible outcomes $o_s \in \mathcal{O}$. Service selection deals with finding the service s where o_s is optimal according to user's preferences.

In this context a preference structure is defined by the complete, transitive, and reflexive relation $\succeq$. E.g. the service $o_1 \in \mathcal{O}$ is preferred to $o_2 \in \mathcal{O}$ if $o_1 \succeq o_2$. The preference structure can be derived from the value function $v^i(o)$ of a user i.

$$\forall a, b \in \mathcal{O} : o_a \succeq o_b \Leftrightarrow v^i(a) \geq v^i(b) \qquad (1)$$

The function $v^i(a)$ represents the utility defined by the relation $\succeq$ in a sense that the attribute values can be ranked by comparing the numeric values of the value function. Basically, we could also perform decision making directly in terms of preference orders and avoid the concept of value functions, but multi-attributive utility theory offers advantages regarding compactness and analytic manipulability [21]. It allows to decompose complex outcome spaces into complex utility functions composed of several lower-dimension functions. Thus, we can describe the preference structure for the attributes relevant to a specific service separately and then combine them using a complex utility function.

Based on these definitions the valuation of a service depends on the preference structure of a user i derived from the value function of the attributes X. Hence, the overall valuation can be approximated by using the following additive value function.

$$V^i(x) = \sum_{j=1}^{n} \lambda_j^i v_j^i(x_j) \qquad (2)$$

For the additive value function above we assume mutual preferential independence between the attributes [12]. Under this assumption we can easily aggregate the utility functions $v_j^i(x_j)$ of the individual attributes j to obtain the overall valuation of a service. However, additive value functions are valid in many real world scenarios and might still provide a good approximation, even when mutual preferential independence does not hold exactly [16]. The weighting factor λ_j^i is normalized in the range $[0, 1]$ and allows to model the relative importance of an attribute j for a specific agent i.

In the following section we describe an approach to approximate the preference structure of a user based on the polices that are applied in the system. Knowing this structure we can not only determine if a service complies with the polices, but we can also estimate the degree of compliance. Based on the individual value function of the attributes, the overall valuation of service can be derived by applying equation 2. Because we only consider the service valuation of one (and the same) user we omit the parameter i in the following considerations.

[5]We can treat a range of numeric values as a finite vector of discrete values or a infinitely long vector representing a continuum of values [11, 3]

6 Preference Elicitation from Policies

As discussed above, policies guide the behavior of a system by constraining attributes. Thus, policies define preference relations between the attribute values x_{jk}. We suggest to analyze these relations in order to generate a value function v_j for the attribute X_j. This allows to make the step from a pure boolean-centered view that is usually adopted when talking about policies to a more quantitative perspective. Doing this helps us to draw conclusions not only about the suitability of a service, but also about the influences of policy violations on the user's service valuation.

Firstly, we introduce the concept of *value function patterns* and describe how such patterns can be deduced from policies (Section 6.1). Secondly, it is outlined how value function patterns can be combined (Section 6.2). Finally, we address the estimation of the free parameters in such patterns (Section 6.3).

6.1 Value Function Patterns

Looking at the individual attributes x_j, a policy could have significant influences on the characteristics of the value function v_j. As mentioned above, our policy description framework allows to express a limited number of policy classes in a formal and explicit way. Each of these policy classes indicates a preference structure with specific characteristics. We represent these characteristics by *value function patterns*. Value function patterns are predefined, parameterized utility functions which have fixed characteristics, but the concrete valuations are parameterized.

In the following we present for each policy class the corresponding utility function pattern. Afterwards, these patterns can be adapted to get a concrete value function by determining the free parameters.

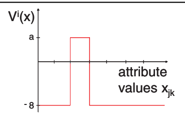
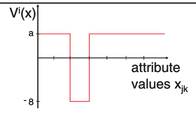
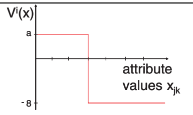
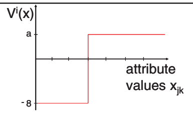
	nominal scale	ordered scale
$==$		
$\neq$		
$\leq, <$		
$\geq, >$		

Figure 4. Value Function Patterns for Obligations

Obligation: Obligations define requirements that have to be met in any case. This means, an attribute j must be valued by the attribute value $x_{jp} \in \mathcal{P}_j$. Values nearby are considered as policy violation.

In case of an ordered scale (i.e. ordinal, interval, or ratio scale) we also have to consider the operators such as *greater than*, *smaller than*, etc. That means a policy could also state that only attribute values *greater than* x_{jp} are valid. In this case the Region $\mathcal{P}_j$ is defined by $\mathcal{P}_j = \{x_{jk} | \forall k > p\}$, i.e. all x_{jk} with $k > p$ are valid. Other operators can be handled similarly by redefining the sets $\mathcal{P}_j$ and $\mathcal{W}_j$.

Due to the fact that a service must not be selected if an obligation is violated, the function value has to be set to $v_j(x_{jk}) = -\infty$ in this case. Otherwise, i.e. the obligation is met by the service, the function is valued by the parameter a. The following function represents the value function pattern for a policy that requires an attribute j to be valued by $x_{jk} \in \mathcal{P}_j$.

$$v_j(x_{jk}) = \begin{cases} -\infty & x_{jk} \notin \mathcal{P}_j \\ a & x_{jk} \in \mathcal{P}_j \end{cases} \tag{3}$$

In case of a negative obligation the inverted policy operator has to be applied (e.g. '$\neq$' instead of '$=$'). Thus, we simply replace the definition of the sets $\mathcal{P}_j$ and $\mathcal{W}_j$ by each other. For instance, an attribute X_j *must not* be valued by x_{jp} redefines the sets in the following way: $\mathcal{P}'_j = \{x_{jk} | k \neq p\}$ and $\mathcal{W}'_j = \{x_{jk} | k = p\}$. Consequently, we reuse the formula 3 with the revised sets $\mathcal{P}'_j$ and $\mathcal{W}'_j$, respectively. Both options are visualized in figure 4.

Recommendation: Recommendations introduce some fuzziness, i.e. a policy violation of a service does not lead directly to the disqualification of the service, but to a lower valuation by the agent. Therefore, the utility will never be $-\infty$. For defining the value function patterns we have to distinguish between nominal, ordinal and continuous scales.

In case of a nominal scale we get two utility levels: The parameter a represents the valuation if the policy is not violated, and b the valuation in case of a violation. For a policy stating that the attribute X_j should be valued by x_{jp}, $\mathcal{P} = \{x_{jk} | \forall k = p\}$, the value function pattern is defined similarly to equation 3:

$$v_j(x_{jk}) = \begin{cases} a & x_{jk} \in \mathcal{P}_j \\ b & x_{jk} \notin \mathcal{P}_j \end{cases} \tag{4}$$

Having an ordered scale, i.e. ordinal, interval, or ratio, the function gets more complex, because the utility decreases with the degree of the policy violation. This rate is determined by the 'distance' between the attribute values $x_{jp} \in \mathcal{P}_j$ expected by the policy and the real value $x_{jk} \in \mathcal{X}_j$ in a system.

Ordinal scales do not contain any information about the distances between the categories. Thus, we have to assume that all distances are equal and the only information available is the ranking of the categories. From policies we can acquire information about which end of the scale contains attribute values with high valuation and which contains the values with low valuation. Therefore, we first have to determine the next *valid* attribute value $x_{j\bar{p}}$ of x_{jk}.

$$x_{j\bar{p}} = \begin{cases} \min(\mathcal{P}) & \text{if } k < \bar{p} \wedge x_{j\bar{p}} = \min(\mathcal{P}) \\ \max(\mathcal{P}) & \text{if } k > \bar{p} \wedge x_{j\bar{p}} = \max(\mathcal{P}) \end{cases} \tag{5}$$

Then, we get the following function for the attribute values x_{jk}. In case there is no policy violation we get a utility of a otherwise we have a valuation between 0 and a depending on the distance to a valid attribute value $x_{j\bar{p}}$.

$$v_j(x_{jk}) = \begin{cases} \frac{a-b}{\bar{p}-1}k, & \text{if } k - \bar{p} < 0 \\ a, & \text{if } x_{jk} \in \mathcal{P} \\ \frac{a-b}{m-\bar{p}}(m-k), & \text{if } k - \bar{p} > 0 \end{cases} \tag{6}$$

Continuous ratio and proportional scales with range $[min_j, max_j]$, in contrast, contain concrete information about the distance of two attribute values that can be used to determine the valuation. Therefore, we have to redefine equation 6 in the following way:

$$v_j(x_{jk}) = \begin{cases} \frac{a-b}{x_{j\bar{p}}-min_j}x_{jk}, & \text{if } x_{jk} < \min(\mathcal{P}) \\ a, & \text{if } \max(\mathcal{P}) > x_{jk} > \min(\mathcal{P}) \\ \frac{(a-b)}{max_j - x_{j\bar{p}}}(max_j - x_{jk}), & \text{if } x_{jk} > \max(\mathcal{P}) \end{cases} \tag{7}$$

As you can see in equation 6 ordinal scales (with discrete values) are represented by step curves, whereas ratio and proportional scales can be represented by continuous curves. For the sake of compact presentation we do not distinguish between the two types of scales in figure 5. Never-

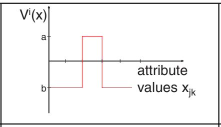
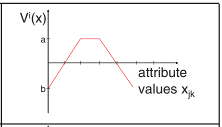
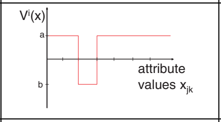
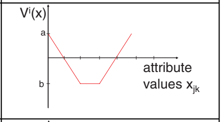
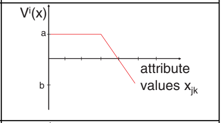
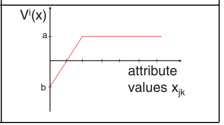
	nominal scale	ordered scale
$==$		
$\neq$		
$\leq, <$		
$\geq, >$		

Figure 5. Value Function Patterns for Recommendations

theless, one should be aware that between the different ordered scales (ordinal, interval, and ratio) the patterns differ slightly.

6.2 Combination of value function patterns

In the previous section it is outlined how we can obtain value function patterns for an attribute based on single policies constraining this attribute. In this section we deal with the fact that there might be more than one policy about the same attribute. Therefore, we derive several different patterns describing the agent's preference structure of the attribute. Figure 6 shows an example for such a scenario. Additionally, there could be already existing value functions about an attribute not derived from policies that also have to be considered when making a decision (e.g. background knowledge). In order to get one integrated value function pattern that represents the different sources we need to find a way to aggregate the different patterns.

As mentioned in section 4, the ontology supports the integration of several policies into collections by relating them with boolean and implication operators. These logical relations between the policies we can use for the aggregation of the value function patterns. Policies $1 \dots q$ with the corresponding value functions $v_{j,1} \dots v_{j,q}$ and the collection operators *op* valued by '$\wedge$' for a logical and-relation and '$\vee$' for a logical or-relation can be combined by the following formula:

$$v_j(x_{jk}) = \begin{cases} max(v_{j,1}(x_{jk}), \dots, v_{j,q}(v_{jk})) & \text{if } op = `\vee' \\ min(v_{j,1}(x_{jk}), \dots, v_{j,q}(x_{jk})) & \text{if } op = `\wedge' \end{cases} \quad (8)$$

In case of an collection with a logical and-relation we take the minimal value to make sure that a violated obligation (valued by $-\infty$), cannot be overruled by another policy. In case of a logical or-relation we take the maximal valuation of an attribute value x_{jk}, because here we can choose freely between the alternatives. Of course, depending on the scenario also other aggregation functions might be useful.

Implications allow to express dependencies between policies, e.g. messages have to be encrypted with a 128 bit key if they contain sensitive data (*data* = 'sensitive' $\rightarrow$ *key length* = 128). In this context, we have to distinguish between the *head* and the *body* of the implication. Only the policy statement in the body contains information about the preference structure of an attribute and thus, indicates a value pattern. The head is only used to define in which case this pattern should be considered. In our framework this can be modelled by a simple rule that states if a pattern should be considered or not.

6.3 Parameter Estimation

In this section we outline how the parameters of the utility function patterns can be determined. Thereby, we want to minimize the amount of additional information gathered from the user of the system directly, because we think complex and time-consuming customization of the system is a major obstacle regarding user adoption.

Therefore, we strongly limit the scope of requests to the following questions:

(1) What is the willingness to pay v_{opt} of the desired service in case all requirements are met by the service? While answering such a question is difficult for users, defining an upper bound is absolutely necessary in real-world scenarios. It is very doubtful that human users will hand over their decisions to autonomous machines without defining an upper limit for the costs that could arise. We can regard this also as a mandatory policy about the attribute 'price'. v_{opt} is typically expressed in monetary units.

(2) What is the relative importance w_j of an attribute j? Such information can be gathered efficiently by using a taxonomy that organizes the attributes of a service. To minimize the user input we can ask for the weights on a higher level in the taxonomy and then propagate this information down to the leaves. We call these more general attributes that are not directly attributes of the service, but rather classify the service attributes, *meta-attributes*. Using this meta-attributes provides several advantages: It reduces the number of attributes that must be weighted by the user; it is a lot easier for the user to judge these general attributes; the information gathered is basically more general and thus, it could be used for a greater variety of service types.

In order to predict the exact values of the parameters we have to make some simplifying assumptions. To represent the valuations of the individual attributes we use the range $v_j \in [0, 1]$. We assign the full willingness to pay, i.e. $v_j(x_{jp}) = 1$, to the most preferential attribute value x_{jp} and $v_j(x_{jk}) = 0$ to the least preferential attribute value x_{jk} that is not forbidden by an obligation (in this case it is $-\infty$). Under this assumption we can easily define the parameters a and b in our value function patterns, because a represents always the most preferential attribute value and b the least preferential.

$$a = 1 \tag{9}$$

$$b = 0 \tag{10}$$

Secondly, we assume that all attributes contribute to the value of a service according to their relative importance, i.e. we can calculate a normalized weighting factor $\lambda_j = \frac{w_j}{\sum_j w_j}$ that represents the proportion of the overall willingness to pay contributed by an attribute j. The proportional result $V(x) = \sum_{j=1}^{n} \lambda_j v_j(x_j)$ can be easily transformed to a monetary value ϑ by using the following formula:

$$\vartheta = v_{opt} * V(x) \tag{11}$$

Based on the functions above service offers can be analyzed and compared regarding the utility they provide to the customer.

7 An Example

In this section we outline a concrete example of our approach. Therefore, we consider the following four policies which all have to be met by the selected service:

- *P1: Invoke only services that are ISO 9000 certificated.*
- *P2: Only services which have a credit rating of A and B should be invoked.*
- *P3: Generally, all outgoing messages have to be encrypted with a key length of at least 64 bit.*
- *P4: A key length should be at least 128 bit if the payment method is credit card.*

Specification of policies. Policy *P1* constrains the attribute *Certificate* which belongs to the meta-attribute *QoS*. The attribute has a nominal scale and can be valued by {*'ISO', 'TUEV', 'DNV', 'other', 'non'*}. The policy states that the attribute *Certificate* must be valued by *'ISO'* (Operator: =). That means in this case the set $\mathcal{P}_1$ is defined by $\mathcal{P}_1 = \{\text{'}ISO\text{'}\}$.

Policy *P2* gives a recommendation about the credit rating of a service provider. The attribute *CreditRating* is valued by an ordinal scale with the attribute values {*'A', 'B', 'C', 'D', 'E'*}. All values *'A'* and *'B'* are considered as unproblematic and form the set $\mathcal{P}_2$. But because the policy is only

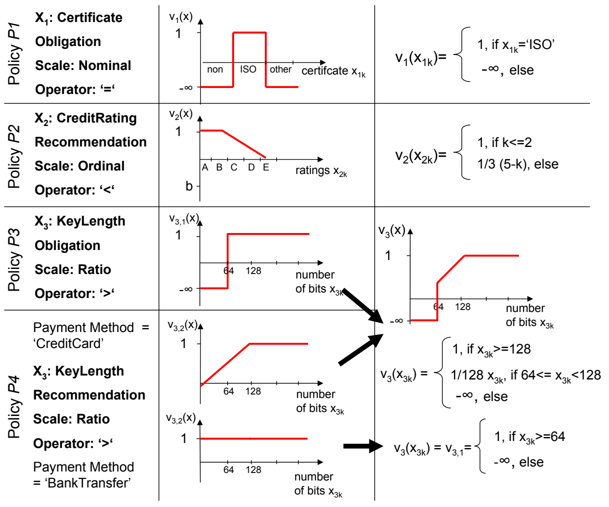


Figure 6. Example: Derived value functions

a recommendation, a service with a credit rating below *'B'* might also be selected (especially, if the relative importance of *CreditRating* is very low).

P3 says that all considered services must support a key length of at least 64 bit, i.e. the attribute *KeyLength* is valued by using a ratio scale with the numeric range [0,2048]. Therefore, the set of valid attribute values is defined by $\mathcal{P}_{3,1} = \{64, \ldots, 2048\}$.

The fourth policy is a recommendation that combines two attributes: *KeyLength* and *PaymentMethod*. A *KeyLength* above 128 bit will be preferred if *PaymentMethod* is valued by *'CreditCard'*, otherwise all key lengths are allowed. For *PaymentMethod* only two options are possible, *'CreditCard'* or *'BankTransfer'*. Thus, we have a nominal scale here and the set $\mathcal{P}_4 = \{$'Credit Card', 'PaymentMethod'$\}$. There is already a constraint on the attribute *KeyLength* and we have to generate a second one that results in the set $\mathcal{P}_{3,2} = \{128, \ldots, 2048\}$. This constraint has to be considered only if one pays with credit card.

Identifying value function patterns. Policy aspects that are relevant for classification and thus, for determining the value patterns are summarized in figure 6. The policies above indicate preference structures for the attributes *Certificate*, *CreditRating*, and *KeyLength*. We do not have any information about the preference structure of attribute *PaymentMethod*. By analyzing the policy class defined by the policy constraint, scale, and modality we can identify a suitable value function pattern reflecting the policy statement. As mentioned above the attribute *Certificate* is valued by an nominal scale and the policy *P1* is an *obligation towards* the task *invoke*. Thus, we have to chose the first pattern of figure 4, i.e. the only attribute value with an valuation that is not $-\infty$ is *'ISO'*.

The attribute *CreditRating* is valued by an ordinal scale.

The policy *P2* is only a recommendation and uses the comparison operator '$<$'. Therefore, we have to chose in figure 5 the third pattern in the right column.

The third attribute *KeyLength* uses a ratio scale and is addressed by two policies. *P3* is a simple obligation with a '$>$'-operator. This leads to the third pattern in the right column of table 4. *'P4'* constrains the attribute *KeyLength* with a recommendation that uses the '$>$'-operator in case *PaymentMethod = 'CreditCard'*. Thus, if this is the case we have to use the last pattern represented in column two of figure 5. For the additive value function (equation 2) we need one integrated pattern for each attribute. For attribute *KeyLength* ($j = 3$) we have two patterns and both of them have to be met. Therefore, we have an '*all*'-operator in terms of WS-Policy that connects them. As outlined in section 6.2, combination of patterns can be done by using the formula 8: $v_3(x_{3k}) = min(v_{3,1}(x_{3k}), v_{3,2}(x_{3k}))$. The result of this 'merge' you can see in figure 6.

Estimating the free parameters. In order to adapt the patterns to the concrete policy examples, information about the relative importance w_j of the attributes and the price v_{opt} an agent is willing to pay for a service with a optimal set of attribute values is required. Under the assumptions mentioned in section 6.3 we can use the equations 9 and 10 to set the parameters a and b: $a = 1$ and $b = 0$. Here, we assume a valuation for the optimal service of $v_{opt} = 80$ and a weighting vector for the attributes *Certificate*, *CreditRating*, and *KeyLength* of $w = (40, 40, 20)$. Consequently, we get normalized weights of $\lambda = (0.4, 0.4, 0.2)$. The formulas we get after inserting the parameters a and b are shown in figure 6.

The overall valuation of a service can be approximated by the additive value function, where $\vec{x}$ represents an offered service.

$$V(x) = 0.4 * v_1(x_1) + 0.4 * v_2(x_2) + 0.2 * v_3(x_3) \quad (12)$$

This means, for two offers x^1 = *('ISO','A', 128, 'CreditCard')* and x^2 = *('ISO','C', 64, 'BankTransfer')* the valuations would be $V(x^1) = 0.4 * 1 + 0.4 * 1 + 0.2 * 1 = 1$ and $V(x^2) = 0.4 * 1 + 0.4 * \frac{2}{3} + 0.2 * 1 = \frac{13}{15}$, respectively. Hence, the service with offer x^1 should be preferred. In case monetary valuations are needed, formula 11 can be used: This results in $\vartheta^1 = 80$ and $\vartheta^2 = 80 * \frac{13}{15} = 69.33$.

Figure 6 summarizes the example by showing graphically how a policy statement can be used to derive a value function.

8 Related Work

Many policy languages such as WS Policy, WS Security, EPAL, XACML, and others emerged in the SOA community. Our work differs from these languages in that we base on a clean and extensible conceptual model. Furthermore, the WS* languages base on discrete reasoning or only vaguely define the semantics.

Like our work, KAoS [20] and Rei [10] are also based on formal ontologies. However, both apply a discrete reasoning approach that allows for pure boolean decisions only and do not aim at unifying policy languages via foundational ontologies.

In economics, similar work is done in the area of preference elicitation [1] and utility analysis. The goal here is to collect information about the user's preference structure in order to support the system in making the appropriate decisions. We suggest to harvest this information from policies, which seems like a natural way for users to express their preferences. Obviously, a combination of different methods is feasible here. Additionally, in multi-attribute utility theory alternative models for expressing utility functions can be applied as done in [11], for instance.

Current approaches in service selection usually focus on quality of service and apply community-based approaches by introducing reputation systems as done in [18, 2, 14]. Our approach, in contrast, extends service selection beyond QoS-attributes and approximates the user's preference structure explicitly without the need of community data.

In policy-based systems often hybrid approaches are applied, where in a first step policies are enforced to determine which services can be invoked at all. In a second step, application specific concerns are handled by continuous valuations. Our approach allows a single step decision process, thus, enabling obligations and recommendations to be included in the decision directly.

9 Conclusion and Outlook

We presented a utility based approach for policy driven service selection. Our work addressed several major problems of today's policy based systems, namely the heterogeneity of policy languages, the problem of combining technical and application specific attributes, and the challenge to coming up dynamically with a ranking and valuation of the available services. Drawing from a wide range of fields, our approach fits into service oriented architectures, integrates existing policy languages seamlessly, and allows the user to specify his valuation in a natural way via policies. Although the approach is applied to service selection in this paper, it can be applicable to various other scenarios.

We are going to extend our framework in the following directions. We are currently in the process of developing a prototype system and plan to evaluate the approach by feeding policies into the system and having users verify the calculated utility manually. From a methodological point of view we plan on relaxing some of our assumptions such as the attribute independence. We are also going to extend the expressiveness of the ontology (e.g. priorities for policies) and refine the value function patterns. Last but not least, our approach can be extended to the provider side and can be used for policy debugging.

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 SESAM Internetökonomie project of the Federal Ministry of Education and Research (BMBF), and the EU in the IST project SEKT (IST-2003-506826).

References

[1] Li Chen and Pearl Pu, *Survey of preference elicitation methods*, Technical Report IC/200467, Swiss Federal Institute of Technology in Lausanne (EPFL), Lausanne, Switzerland, July 2004.

[2] J. Day and R. Deters, *Selecting the best web service*, Proc. of 14th Annual IBM Centers for Advanced Studies Conference (CASCON), 2004, pp. 293–308.

[3] P. Faratin, M. Klein, H. Sayama, and Y. Bar-Yam, *Simple negotiating agents in complex games*, Proc. of 8th Int. Workshop on Intelligent Agents VIII, Springer-Verlag, 2002, pp. 367–376.

[4] P. Faratin, C. Sierra, and N.R. Jennings, *Negotiation decision functions for autonomous agents*, Int. Journal of Robotics and Autonomous Systems (1997), 159–182.

[5] P. Feltovich, J.M. Bradshaw, R. Jeffers, and A. Uszok, *Order and KAoS: Using policy to represent agent cultures*, Proc. of AAMAS 03 Workshop on Humans and Multi-Agent Systems (Melbourne, Australia), 2003.

[6] A. Gangemi, *Ontology design patterns*, Tech. Report 2004#1, Laboratory for Applied Ontology, 2004.

[7] A. Gangemi and P. Mika, *Understanding the semantic web through descriptions and situations*, Confederated Int. Conf. DOA, CoopIS and ODBASE, LNCS, Springer, 2003.

[8] 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.

[9] W.N. Hohfeld, *Some fundamental legal conceptions as applied in judicial reasoning*, Yale Law Journal **23** (1913).

[10] 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.

[11] A. H. Karp, *Representing utility for automated negotiation*, Technical Report HPL-2003-153, HP Laboratories Palo Alto, July 2003.

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

[13] 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, 2004.

[14] Y. Liu, A. H. Ngu, and L. Z. Zeng, *Qos computation and policing in dynamic web service selection*, WWW Alt. '04: Proc. of 13th Int. WWW Conf. on Alternate track papers & posters, 2004, pp. 66–73.

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

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

[17] M. Sergot, *A computational theory of normative positions*, ACM Transactions on Computational Logic **2** (1002), no. 4, 581–622.

[18] R.M. Sreenath and M.P. Singh, *Agnet-based service selection*, Journal of Web Semantics **1** (2004), no. 3.

[19] S. Staab and R. Studer, *Handbook on ontologies*, Springer Verlag, Heidelberg, 2004.

[20] A. Uszok, J. M. Bradshaw, R. Jeffers, A. Tate, and J. Dalton, *Applying KAoS services to ensure policy compliance for semantic web services workflow composition and enactment.*, Int. Semantic Web Conf. (ISWC'04), 2004, pp. 425–440.

[21] M. P. Wellman and J. Doyle, *Modular utility representation for decision-theoretic planning*, Proc. of 1st Int. Conf. on AI Planing Systems 1992 (AIPS-92), June 1992, pp. 236–242.