

Available from: http://dx.doi.org/10.1109/CCGrid.2011.36

Copyright © 2011 IEEE.

This is the author’s version of the work. It is posted here with the permission of the publisher for your personal use. No further distribution is permitted. If your library has a subscription to these conference proceedings, you may also be able to access the published version via the library catalogue.
A Flexible Policy Framework for the QoS Differentiated Provisioning of Services

Mohan Baruwal Chhetri  
Center for Complex Software Systems and Services  
Swinburne University of Technology  
Hawthorn, Australia  
mchhetri@swin.edu.au

Bao Quoc Vo  
Center for Complex Software Systems and Services  
Swinburne University of Technology  
Hawthorn, Australia  
bvo@swin.edu.au

Ryszard Kowalczyk  
Center for Complex Software Systems and Services  
Swinburne University of Technology  
Hawthorn, Australia  
rkowalczyk@swin.edu.au

Abstract—We propose a policy-based framework for the QoS differentiated provisioning of services. The proposed framework improves the state-of-the-art in policy-based preference specification by combining cardinal and ordinal preferences. We describe the underlying models, focusing on the key features and contributions of the proposed framework. We also show how, using our framework, the QoS evaluation problem can be translated to a Constraint Satisfaction Problem while preserving the semantics of the preference policies.

Keywords-QoS preference model, attribute-value assertion, utility-value assertion, conditional attribute-value assertion

I. INTRODUCTION

Quality of Service remains a key differentiator in an increasingly competitive services market where a number of providers can offer functionally equivalent services. If service providers are to remain competitive, they have to show maximum flexibility in the offered service quality levels so that they can provide their service to a diverse set of consumers. The commonly used offers-based approach\(^1\) is not suitable for this. A better alternative for service providers is to specify their preferences and constraints over all possible service configurations i.e. the entire configuration space. Service consumers can compose dynamic requests based on their specific requirements, which can be evaluated against the provider preferences to determine whether the requested service quality can be offered or not. To do this both service providers and consumers require a QoS preference model that is sufficiently expressive, highly flexible, easy to use, and supports varying levels of complexity in representing preferences. Another key requirement is the automation of the QoS evaluation and decision-making process. This can result in significant time-savings and cost and effort reductions, and also enables flexible adjustments to preferences in response to dynamic business conditions.

To address these challenges we propose a policy-based framework to enable the QoS-differentiated provisioning of services. Policies provide a natural and intuitive way of representing QoS requirements, capabilities and preferences over them. Preference policies can capture the human-defined preferences over the QoS capabilities and requirements. These policies can then be interpreted by an autonomous policy engine which can reason about the preferences, evaluate incoming service requests, generate offers and counter offers, and if feasible negotiate an agreement. The use of a declarative, machine interpretable formalism enables automated decision making in conformance with the human counterpart. Furthermore, separation of concerns is possible since policies state what the QoS preferences are but not how the QoS evaluation should be done. This facilitates manageability while providing a high degree of flexibility.

A number of proposals for policy-based QoS preference specification have been proposed in the literature \([1][2][8][9][10]\). Some of them use ad-hoc approaches with no strong theoretical foundations, while others have limitations and are not easy to use as shown in Section III. We use the theory of utility functions, which offers the most expressive form of preference specification and has very strong theoretical properties \([11][12]\). The utility function is an objective function that assigns a scalar value to all possible combinations of attribute-value assignments i.e. all possible service configurations. A key benefit of using utility functions for specifying QoS preferences is that it enables the on-the-fly evaluation of each and every service request. We propose a flexible QoS preference model based on utility theory and utility functions. The key contributions of our proposed framework can be summarized as follows:

- We introduce the utility-value assertion, which, when combined with a comprehensive cardinal utility function\(^2\) is sufficient by itself for specifying preferences over the entire configuration space.
- We allow service providers to combine cardinal and

\(^1\)In the offers-based approach the service provider advertises a set of predefined service configurations that the consumer can choose from. A service configuration is an assignment of values to each QoS attribute. The service consumer is restricted to choosing from these predefined configurations and cannot compose dynamic service requests because of which the provider cannot learn about the consumer requirements and preferences.

\(^2\)A comprehensive cardinal utility function covers the entire configuration space and is able to assign a numeric value or score to every possible service configuration.
ordinal preferences so that they can show maximum flexibility in the offered service levels. If the cardinal utility function is not comprehensive (which is generally the case) and covers only a subset of the configuration space, service providers can use additional attribute-value and conditional attribute-value assertions to specify ordinal preferences\(^3\) over the remaining configuration space thereby maximizing the chances of forming agreements.

- Using the proposed QoS preference model, we show how the QoS evaluation problem can be translated to a Constraint Satisfaction Problem (CSP) while preserving the semantics of the preference policies.

To the best of our knowledge, our model is the first to introduce the utility-value assertion and combine cardinal and ordinal preferences. This allows both domain experts as well as naive users to express their requirements, capabilities and preferences over the QoS attributes.

The rest of this paper is organized as follows. Section II describes a simple motivating scenario which is used in Section III to highlight the limitations of existing approaches for policy-based preference specification. Section IV illustrates our proposed approach to capture QoS preferences more comprehensively through the use of simple examples. We present our formal preference model in Section V and in Section VI we show how our model can be translated into a CSP. In Section VII we discuss the benefits of our approach and potential future work. Section VIII concludes the paper.

II. MOTIVATING SCENARIO

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Measurement Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Time (ET)</td>
<td>Milliseconds (ms)</td>
</tr>
<tr>
<td>Availability (A)</td>
<td>No Unit</td>
</tr>
<tr>
<td>Throughput (T)</td>
<td>Transactions per second (tps)</td>
</tr>
</tbody>
</table>

To motivate our work, we consider the simple scenario of a service which has three customizable QoS attributes - *Execution Time*, *Availability* and *Throughput*. The service provider can offer the service at different QoS levels to its consumers based on its own preferences and the individual consumer requirements. All valid service requests and offers are potential service configurations. The service request can either be a partial service configuration (i.e. it includes only some of the QoS attributes) or a complete service configuration (i.e. it includes requirements on all the QoS attributes). The service configuration that is acceptable to both the service consumer and the service provider is captured in the Service Level Agreement (SLA). There are different ways in which a service provider or consumer can express preferences over the QoS attributes. Based on these preferences, the autonomous decision maker determines whether an incoming service request or offer is acceptable or not.

III. LIMITATIONS OF EXISTING APPROACHES

There are several proposals on policy-based approaches for preference specification [1][2][8][9][10]. In this section we systematically present the different approaches, using them to model the QoS preferences for the scenario presented in the Section II and discuss their shortcomings.

A. Using attribute-value assertions

In this approach policy-authors specify their preferences over non-functional attributes using only attribute-value assertions. These assertions are for single attributes and collectively can be used to classify a given alternative as either being acceptable or not-acceptable [1][2]. Different types of mathematical operators such as $=$, $<$, $>$, $\geq$, and/or semantic relationship operators such as *atMost*, *atLeast* and *around* can be used to make more sophisticated and flexible preference statements [4]. This allows the ranking of different alternatives on the basis of individual preferences, from which an overall ranking can be computed. However, this approach has limitations as shown in Example 1 and Example 2.

**Example 1: Attribute value assertions (Using single value assignments)**

Let us consider a simple example where the service requester has strict preferences over *Execution Time*, *Availability* and *Throughput* expressed using attribute-value assertions:

$P_R = (ET = 10ms, A = 0.975, T = 10tps)$

Let us assume that there are three different predefined service configurations advertised by the service provider as shown below:

<table>
<thead>
<tr>
<th>Offer</th>
<th>Service Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_1$</td>
<td>$(ET = 10ms, A = 0.985, T = 15tps)$</td>
</tr>
<tr>
<td>$O_2$</td>
<td>$(ET = 8ms, A = 0.985, T = 12tps)$</td>
</tr>
<tr>
<td>$O_3$</td>
<td>$(ET = 7ms, A = 0.99, T = 9tps)$</td>
</tr>
</tbody>
</table>

Using pure syntactic matching, none of the offers are acceptable, although manual examination shows that $O_1$ and $O_2$ are both acceptable offers.

**Example 2: Attribute value assertions (Preferences over ranges)**

Let us modify the service requester’s preferences using ranges:

$P'_R = (ET \leq 10ms, A \geq 0.975, T \geq 10tps)$

Using $P'_R$, it is possible to determine that offers $O_1$ and $O_2$ are acceptable.

From $P'_R$, it is clear that a lower *Execution Time*, higher *Availability* and a higher *Throughput* is preferred by the service requester. However, ranking the offers based on individual preference statements results in the following decision matrix where it is not possible to choose a better offer due to a tie in the scores.

<table>
<thead>
<tr>
<th>Offer</th>
<th>$ET$</th>
<th>$A$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_1$</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$O_2$</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

\(^3\)Ordinal utility allows the ranking or ordering of different service configurations even though they don’t assign numeric values to them.
B. Using attribute-value & attribute-importance assertions

Since using attribute-value assertions alone has its limitations, an improvement is to allow policy-authors to specify additional attribute-importance assertions by which attributes can be assigned a rank or a relevance score. It is natural for users to make statements such as A is more important than B or A is of the same importance as B. While the use of relative importance statements does provide additional preference information and can be useful in evaluating and ranking alternatives, it also has limitations as shown in Example 3.

<table>
<thead>
<tr>
<th>Example 3: Attribute value and Attribute importance assertions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let us extend Example 2 by adding an attribute-importance assertion.</td>
</tr>
</tbody>
</table>
| \[
| P_{R}'''' = (\text{ET} \leq 10\text{ms}, A \geq 0.975, T \geq 10\text{tps}, \text{ET is more important than T})
| \]
| Using the decision matrix shown in Example 2, we can consider O_2 to be better than O_1 since 
| ExecutionTime is more important than Throughput. However, if both attributes are equally important, then, again there is a tie. From the preference policy, there is no way to determine which offer is better because we don’t know the relationship/mapping between 1ms and 1tps. |

Example 4. Attribute value assertions, Attribute importance assertions and value function patterns

Let us modify Example 3, so that the service requester specifies explicit values for the attribute-weights. \[
| P_{R}'''' = (ET \leq 10\text{ms}, A \geq 0.975, T \geq 10\text{tps}, W_{ET} = 0.4, W_{A} = 0.2, W_{T} = 0.4) |
| Since all three attributes are monotonic and take continuous values we can approximate their utility functions to be Linear and compute the total utility using the weighted sum. |
| For availability and ExecutionTime the upper and lower bounds can be estimated from \( P_{R}'''' \). For Throughput, the lower bound is 10tps. Let us assume that the upper bound is 25 tps as specified by the domain expert. |
| Using this approximation, and using the weighted sum, the utility of O_1 in Example 1 is 0.132 and that of O_2 is 0.1332 implying that O_2 is better than O_1. |
| While this approach permits the evaluation and ranking of offers, the major limitation is that it restricts the utility function definition to a limited set of predefined functions which may or may not capture the actual preferences of the policy-authors. |

C. Using attribute-value & attribute-importance assertions with utility function patterns

In this approach [8][9] policy authors specify preferences through conditional and unconditional attribute-value assertions. They also have to provide attribute-importance statements. From these assertions, the corresponding value functions are estimated as shown in Example 4. The authors in [8][9] have defined a fixed set of value functions - point based functions, piecewise linear functions and pattern based functions. The utility function for every QoS attribute is estimated from the attribute-value assertions and the total utility is computed using weighted sums.

D. Using utility function assertions

In this approach, policy-authors define the utility by inserting built-in numeric utility functions in the preference rules. In [10], the authors assume that the attributes are preferentially independent, and have a linear utility. The overall utility for a service configuration is then computed as the weighted sum of the individual utilities. We use this approach for our simple example shown in Example 5 where all three attributes have linear utility functions. The difference between this approach [10] and the previous approach [8][9] is that in this approach the utility function definition is explicitly specified in the preference rules as a numeric function. Thus, if the policy language supports a set of built-in numeric functions, policy authors can modify the utility function at any time.

<table>
<thead>
<tr>
<th>Example 5. Attribute value assertions, Attribute importance assertions and utility function assertions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let us extend Example 4, so that the service requester specifies explicit values for the attribute-weights so that it is given by:</td>
</tr>
</tbody>
</table>
| \[
| P_{R}'''' = (\text{ET} = 0\text{ms}, \text{then } u = 1, \text{if } \text{ET} = 10\text{ms}, \text{then } u = 0, \text{if } \text{ET} = 10\text{ms}, \text{then } u = (10-x)/10),
| \text{if } \text{A} = 0.975, \text{then } u = 0, \text{if } \text{A} = 1.0, \text{then } u = 1, \text{if } \text{A} = 1.0, \text{then } u = (40x - 39),
| \text{if } \text{T} = 25\text{tps}, \text{then } u = 1, \text{if } \text{T} = 10\text{tps}, \text{then } u = 0, \text{if } 10 \leq \text{T} \leq 25, \text{u} = (x - 10)/15),
| W_{ET} = 0.4, W_{A} = 0.2, W_{T} = 0.4 |
| Using this definition, the service requester specifies what the upper and lower bounds for each attribute are, what their weights are as well as the numeric function to compute the utility for any intermediate values. The total utility is computed using the weighted sum of the individual utilities. The key limitation of this approach is that it imposes a requirement on the policy language to support complex numeric function definitions. While this particular example uses simple linear utility functions which are reasonably easy to express via policies, representing more complex utility function definitions can make the policy language more complex. |

IV. NEW REPRESENTATION MODEL FOR QoS PREFERENCES

In this section, we present a new representation model for QoS preference specification. We first discuss the concept of utility and how it can be used to capture preferences over QoS attributes. We also discuss how utility functions can be defined through the use of simple examples. We then informally introduce the utility-value assertion and show how it can be used together with attribute-value assertions and conditional-attribute-value assertions to specify cardinal and ordinal preferences. Our approach incorporates most of the existing approaches presented in Section III while extending them. The three main assertions that have been proposed and used in literature are - attribute-value assertions, attribute-importance assertions and utility-assertions. We introduce two new types of assertions - the conditional attribute-value
Availability which has only one customizable QoS attribute or only a part of it. To illustrate this, let us consider a service and it can cover either the entire service configuration space comprehensively and inclusively their utility function is. There are different ways in which the utility function can be defined, which, in the context of a service with multiple QoS attributes, can assign a scalar utility value to every possible service configuration. This is referred to as the utility function $U$ represented as $U : C \rightarrow [0,1]$ which allows for unambiguous and rational decision making, thus facilitating automation of the decision-making process. When service consumers and providers express their preferences using policies, they try to elicit this function $U$. However, in practice, it is seldom possible to define such a comprehensive function and while service providers can use domain experts to define complex utility functions, naive service consumers might not be in a position to define them. Hence it may be beneficial to combine cardinal and ordinal utility.

**A. Utility & Utility Functions**

In Economics [11], utility is defined as a measure of the relative satisfaction from or desirability of consumption of a product or service. It can generally be expressed in two ways - as ordinal utility, or as cardinal utility. Ordinal utility theory states that while the utility of a particular good or service cannot be measured using a numerical scale, different alternatives can be ordered or ranked. Using only attribute-value assertions as discussed in Examples 1, 2 and 3 is one way of expressing ordinal utility. Alternatively, cardinal utility allows the measurement of the strength of preference of a good or service with precision through the use of some objective criteria. Thus, theoretically, an objective function can be defined, which, in the context of a service with multiple QoS attributes, can assign a scalar utility value to every possible service configuration.

This is referred to as the utility function $U$ which is constrained to a subset of the potential assignment space.

### Point-based utility function:

One way to define the utility for Availability is by using the point-based utility function approach as shown in Equation 1. In this example, the point-based utility function assigns utility values for three specific values for Availability. Obviously, this definition does not cover the entire assignment space.

$$U_{\text{Availability}}(a) = \begin{cases} 0.7 & a = 0.8 \\
0.65 & a = 0.65 \\
0.6 & a = 0.6 \end{cases} \quad (1)$$

An alternate numeric function definition from [14] is shown below:

$$U_{\text{Availability}}(a) = K \frac{e^{\alpha (\beta - a)}}{1 + e^{\alpha (\beta - a)}} \quad (3)$$

where $K$ is a normalization factor which is given as:

$$K = (1 + e^{\alpha (\beta - 1)})/e^{\alpha (\beta - 1)} \quad (4)$$

In Equation 4, $\beta$ is the best preferred value for Availability and $\alpha$ is a sensitivity parameter that defines the sharpness of the utility curve. It should be noted that Equation 3 is just one of many possible utility function definitions. As can be seen from Equations 2 and 3, instead of explicitly specifying the utility for specific values of Availability as in Equation 1, a numeric function can be defined which can compute the utility for a number of possible values of Availability. Again, this utility function can be constrained to a subset of all possible values by specifying the upper and lower bounds using constraint operators. As a simple example, the utility function in Equation 2 can be constrained to a subset of the potential assignment space as shown below:

$$U_{\text{Availability}}(a) = \begin{cases} 0.7 & 0.9 \leq a < 0.95 \\
1 & 0.95 \leq a < 1 \end{cases} \quad (5)$$

This particular definition in Equation 5 specifies the cardinal utility for all values above 0.9. However it does not specify the utility for values below 0.9. Hence it cannot be considered as a comprehensive utility function and cannot be used to evaluate service configurations with Availability below 0.9.

Thus, while numeric utility functions and point-based utility functions do provide a very expressive approach to preference specification, they may not cover the entire configuration space and hence may not be sufficient to assign a utility value for every service configuration. We propose combining the numeric utility function with utility-value assertion, attribute-value assertions and conditional attribute-value assertions to provide a more flexible way to express the stakeholder’s utility. This allows policy authors greater flexibility in constraining or expanding the acceptable configuration space.

---

**The attribute-value assertion is a special case of the conditional attribute-value assertion.**
C. Combining assertions

We now show how we can combine different types of assertions to improve the elicitation of utility. For simplicity, we continue with our example of a service with a single customizable QoS attribute Availability. We use the utility-value assertion and the attribute-value assertion separately to illustrate the benefit of complementing numeric utility functions with either of these assertions.

- **Utility-value assertion** - The utility function specifies how the utility is to be computed for a specific configuration and can cover a subset of or the entire configuration space. The utility-value assertion specifies the minimum acceptable utility and constrains the size of the acceptable configuration space by imposing a lower bound on the utility. Let us consider the numeric utility function given in Equation 2 which covers the entire configuration space between 0 and 1. A stakeholder (either the service provider or consumer) can specify the minimum acceptable utility through the utility-value assertion as shown below.

\[ U_{Availability_{min}}(a) \geq 0.6 \]  

(6)

This constrains the acceptable configuration space as shown in Figure 1. The value for the minimum acceptable utility can be changed at any time thereby constraining or relaxing the acceptable configuration space.

- **Attribute-value assertion** - Let us consider the numeric utility function given in Equation 5. It covers only a subset of the entire configuration space. It gives a scalar utility only for values of Availability between 0.9 and 1.0. In such a scenario, the stakeholder can further relax the acceptable configuration space by using a simple attribute-value assertion as shown below:

\[ a \geq 0.8 \]  

(7)

This attribute-value assertion relaxes the acceptable configuration space as shown in Figure 2. The numeric utility function allows computing the utility for all values of availability between 0.9 and 1.0. In addition, the attribute-value assertion states that a service configuration with availability greater than 0.8 is also acceptable even though the exact utility value cannot be computed. Thus, combining the utility-value assertion and attribute-value assertion allows elicitation of both cardinal and ordinal utility.

- **Conditional attribute-value assertion** - When the numeric utility function is defined over a set of QoS attributes and covers only a subset of the potential configuration space, the conditional attribute-value assertion which is of the form if \( p \) then \( q \) can be used to further relax the size of the acceptable configuration space. There are different ways in which the conditional attribute-value assertions can be used.

  - The simplest usage of the conditional attribute-value assertion is to enforce additional constraints under a certain context. It can be described as follows - under a certain context (specified by the condition \( p \)), further constraints (specified by the attribute-value assertion \( q \)) need to be enforced and/or introduced as part of the agreement. A simple example is if \( (RT > 100ms) \) then \( (A \geq 0.99) \) which states that IF the ExecutionTime is greater than 100ms THEN the Availability is required to be at least 0.99.

  - Another usage of the conditional attribute-value assertion is to specify that an attribute-value assertion \( p \) cannot be accepted in general unless it is introduced in a specific context specified by the attribute-value assertion \( q \). A simple example is - if \( (A = 1.00) \) then \( (Premium \geq \$70) \), which implies that Availability of 100% is acceptable ONLY IF the customer is paying a premium of at least $70.

  - A third usage of the conditional attribute-value assertion is for specifying preferences over interdependent QoS attributes. For example, when the ExecutionTime is very high, it does not make sense to have a very high throughput; hence an example conditional attribute-value assertion could be - if \( (ET > 100ms) \) then \( (T \leq 10tps) \); or in the case of a video-on-demand application scenario, if the content is in low-
resolution or audio-only then the connection is preferred to be using shared channels; on the other hand, if the content is high-resolution video, the connection is preferred to be using dedicated channels.

Thus the conditional attribute-value assertion can be used in different ways to specify QoS preferences and to exchange offers and counter-offers.

V. QoS Preference Model

In this section, we first introduce our formal QoS model followed by the assertion model and the policy model. Our QoS preference model allows service consumers sufficient expressivity to formulate service requests using attribute-value assertions and conditional attribute-value assertions. Similarly, service providers can combine cardinal utility functions with utility-value assertions and conditional attribute-value assertions.

A. Formal QoS model

Let us assume that a service has a set of QoS attributes $X = \{x_1, x_2, x_3, ..., x_n\}$. Each attribute takes its value from a finite domain so that $D = \{D_1, D_2, ..., D_n\}$ represents the corresponding set of QoS attribute domains where $D_i$ is the finite set of values that variable $x_i$ can take. The potential configuration space $C$ is given by $C = \{(c_1, ..., c_n) | c_i \in D_i\}$. A utility function $U$ assigns a scalar utility value for every possible service configuration and is given as follows depending upon whether the utility-value assertion is defined or not:

$$ P_{alt} = \begin{cases} \bigwedge_{i \in \{1,...,p\}} A_{a_i} \land \bigwedge_{j \in \{1,...,q\}} A_{c_j} \land A_u, & \text{where } p, q \geq 0 \\ \bigwedge_{i \in \{1,...,p\}} A_{a_i} \land \bigwedge_{j \in \{1,...,q\}} A_{c_j}, & \text{where } p + q > 0 \end{cases} \quad (8) $$

2) Policy: A policy is a collection of policy alternatives which can be combined using the following two policy operators:

- **Any**: which is equivalent to the logical OR construct and enforces the rule that at least one of the listed policies has to be satisfied in order to satisfy the QoS preferences, and
- **ExactlyOne**: which is equivalent to the logical XOR construct and enforces the rule that "exactly one” of the listed policies can be true.

In its normal form, a preference policy can be represented as an enumeration of its alternatives that in turn enumerate each of their assertions. Thus the normal form policy expression is given as:

$$ P_{QoS} = \bigvee_{i \in \{0,...,q\}} P_{alt_i} : \text{Any} \quad (9) $$

where $P_{alt}$ is given by Equation (8) and $q \in \mathbb{N}$ meaning that a policy can have 0 or more alternatives.

Having presented our formal QoS preference model, we next show how we can translate the QoS evaluation problem i.e. the task of evaluating incoming service requests and/or offers, into a CSP while preserving the semantics of the preference policies.

VI. QoS Management Using the QoS Preference Model

One of the key benefits of using a policy-based approach is that it allows separation of concerns i.e. preference policies state what the QoS requirements and capabilities, and, the preferences over them are. However, they do not specify how these preferences are to be used to evaluate service requests and offers and how agreements are reached. This facilitates manageability while providing a great deal of flexibility. In this section, we show one way of applying our models in a practical setting by modelling the QoS management problem as a Constraint Satisfaction Problem. The formulation of the QoS management problem as a CSP allows it to benefit from the strong theoretical foundation that has been built around the constraint programming paradigm [15].
A. Formal Constraint Satisfaction Problem

Formally, a constraint satisfaction problem is defined as a triple \( C = (V, D, C) \) where \( V = \{v_1, v_2, ..., v_n\} \) is a \( n \)-tuple of variables, \( D = \{D_1, D_2, ..., D_n\} \) is the corresponding \( n \)-tuple of domains such that \( v_i \in D_i \) and \( D_i \) is a set of values for \( i = 1, ..., n \). \( C = \{c_1, c_2, ..., c_m\} \) is \( m \)-tuple of constraints which restricts the values that the variables can take. The constraints can be represented as a pair \( (v_i, R_i) \) where \( v_i = (x_{i1}, x_{i2}, ..., x_{il}) \) is a \( l \)-tuple of variables that the constraint applies to, and \( R_i \subseteq \{D_{i1}, D_{i2}, ..., D_{il}\} \) is the set of possible value combinations for these variables. A solution to a CSP is given by the tuple \( s \in D_1 \times D_2 \times ... \times D_n \) that assigns one value to each variable such that all the constraints are satisfied. CSPs can be extended by an objective function that is calculated from the values of the CSP variables and given by \( f : D_1 \times D_2 \times ... \times D_n \rightarrow \mathbb{R} \). The goal is then to find the solution which maximizes or minimizes the objective function. Such a problem is known as a Constraint Optimization Problem (COP).

B. Formal QoS evaluation problem

The QoS management problem can also be formally defined as a triple \( Q = (X, D, P_{QoS}) \) where \( X = \{x_1, x_2, ..., x_n\} \) is a finite set of variables, such that each variable \( x_i \in X \) is associated with a domain of values \( D_i \). The total configuration space is given by \( D_1 \times D_2 \times ... \times D_n \), and \( P_{QoS} \) is the composite QoS preference policy which determines the values that the variables can take. A solution is given by \( s \in D_1 \times D_2 \times ... \times D_n \) that assigns one value to each variable such that all the preferences are satisfied.

C. Mapping QoS management to a CSP

We now describe how we can translate the QoS management problem into a CSP while preserving the semantics of the QoS preference policy. The QoS management problem \( Q \) can be mapped to a CSP as shown below:

\[
Map(V, D, P_{QoS}) = (V, D, Map(P_{QoS}))
\]

where \( Map \) is a generic mapping function. Assuming that the preference policy \( P_{QoS} \) is reduced to compact normal form as shown in Equation (9), it can be mapped to a set of constraint sets as shown in Equation (11) and (12) depending upon the number of alternatives present in the policy.

- If there are no alternatives in the preference policy, then \( P_{QoS} \) maps to an empty constraint set.

\[
Map(P_{QoS}) = \emptyset
\]

- If there are one or more alternatives in the preference policy, then the preference policy maps to a set of constraint sets:

\[
Map(P_{QoS}) = \begin{cases} \bigvee_{i \in \{0,...,m\}} Map(P_{alt_i}) & : \text{Any} \\ \bigcup_{i \in \{0,...,m\}} Map(P_{alt_i}) & : \text{ExactlyOne} \end{cases}
\]

Based on Equations (8) and (14), every policy alternative maps to a set of constraints as shown below:

\[
Map(P_{alt_i}) = \bigwedge_{A_i \in P_{alt}} Map(A_i)
\]

where each assertion \( A_i \) can be mapped to a constraint \( c_i \in C \) as shown below:

\[
Map : A_a \cup A_c \cup A_u \rightarrow C
\]

D. Example Mapping

We now illustrate how our QoS preference model would work in a practical setting. Example 6 shows a numeric utility function definition for our motivating scenario presented in Section II. It also shows how the preferences can be specified by combining the utility-value assertion, the attribute-value assertion and the conditional attribute-value assertion. Using this approximation, the utility of a service offer can be computed using Equation 16. At the same time, the service configuration can be compared against the conditional attribute-value assertions and the utility value assertion.

Example 6. Utility function definition & QoS Preferences

For simplicity, we use the weighted linear utility function. The linear utility function for each attribute is given by:

\[
U(v) = \begin{cases} \frac{(v - v_{\text{min}})}{(v_{\text{max}} - v_{\text{min}})} & \text{for ExecutionTime} \\ \frac{(v_{\text{max}} - v)}{(v_{\text{max}} - v_{\text{min}})} & \text{for Throughput} \\ (1 - v) & \text{for Availability} \end{cases}
\]

The utility function for the service configuration i.e. global utility function, is expressed as the weighted sum of the individual ones

\[
U_g(c) = \sum_{j=1}^{n} w_j . u_j
\]

where \( w_j \) is the weight assigned to each attribute, such that \( w_j \rightarrow [0,1] \) and \( \sum_{j=1}^{n} w_j = 1 \). A service provider can express the QoS preference policy as shown below:

\[
P_{QoS}^{alt} = (U_{\text{min}} = 0.7, A \leq 0.95, T \geq 40\text{tps}) \rightarrow (ET \leq 4\text{ms})
\]

The table above shows how the normative outline of the three types of assertions to be used with WS-Policy as the concrete policy specification language. We show a concrete QoS preference policy using WS-Policy [20] in Example 7.
We also show how the QoS preference policy in Example 7 looks when translated into a Zinc model [16] in Example 8. Zinc is a high-level constraint modelling language used for modelling constraint satisfaction and optimization problems. It is a first-order, functional language with mathematical-like syntax which is extensible and solver-independent. It supports a wide range of different constraint domains and user-defined constraint types. Most importantly a CSP modelled using Zinc is guaranteed to terminate. Since Zinc is solver-independent, the Zinc model can be passed to any one of several back-end solvers such as GeCode [17], ECLiPSe [18], Choco [19] or G12 [21] which give out one or more service configurations which satisfy all the input constraints.

Example 7. Concrete Preference Policy

We have intentionally kept the definition of the numeric functions separate from the preference policies in contrast to the approach in [10] because of the following reasons:

- Inclusion of numeric functions in preference policies makes the policy language complex. Numeric functions can vary from simple mathematical expressions as in Equation (1) to very complex ones (as in Equations (3) and (4)) and providing support for all possible definitions makes the policy language very heavyweight. The idea behind autonomic computing and policy-based computing is to allow policy authors to specify their high-level objectives in a natural, intuitive manner using a simple, expressive and flexible policy language which is lightweight. Hence, in our approach we allow complex utility functions to be defined separately and referred to within the policies.

- It is important to reason about and analyse QoS preference policies. Since attribute-value assertions, conditional attribute-value assertions and utility-value assertion are based on well-defined predicates and logical connectives, a reasoning engine should have no problem reasoning about them. However, reasoning about complex numeric utility functions requires additional capabilities to deal with mathematical expressions and their semantics.
B. Future Work

While combining attribute-value assertions, conditional attribute-value assertions and utility-value assertions allows policy authors greater flexibility in specifying their preferences, it can also lead to potential conflicts. We illustrate this with the use of a simple example based on the scenario in Section II. Let us assume that the domain expert has defined a point-based utility function for a set of six specific service configurations as shown in Equation (17) based on which the service configurations \( c_1, c_3 \) and \( c_4 \) are acceptable.

\[
A_{U_{\min}} = (U \geq 0.7) \quad (17)
\]

Let us now assume that in response to changing business conditions the policy author decides to define an additional conditional attribute-value assertion as shown below:

\[
(A \geq 95) \rightarrow (T \leq 10) \quad (18)
\]

According to the new conditional attribute-value assertion, the service configuration \( c_5 \) is acceptable. However, the utility-value assertion only allows service configurations with a utility greater than 0.7 to be accepted. Thus, we can see through this simple example that potential conflicts can arise when combining utility-value assertions, attribute-value assertions and conditional attribute-value assertions and support is required for detecting and/or resolving such conflicts.

Service providers would benefit from a built-in reasoning mechanism which can detect any conflicting preferences and resolve them. One simple way to avoid such conflicts is to use assertion combining algorithms similar to the rule-combining algorithms such as permit-overrides and deny-overrides in XACML [22]. Such an approach allows policy authors to specify potentially conflicting assertions but then uses rule-combining algorithms to resolve conflicts. While this solution is sufficient for avoiding conflicts, it does not provide any support for conflict detection. Thus one avenue for future research will be to look into the issue of conflict detection and resolution when dealing with QoS preference policies.

Another more interesting area of future research could be looking into how the service provider can negotiate with the service consumers when no solutions exist which satisfy both the consumer request and the provider preferences. Using the CSP formulation, we can check whether a solution exists which satisfies both consumer and provider preferences. But in case there is no acceptable solution, the autonomous reasoner has to be able to reason about its own trade-offs as well as that of the service consumer and try to make or propose tradeoffs. This might mean relaxing the provider side preferences or requesting the consumer to relax its requirements.

VIII. Conclusion

We have presented a policy-based framework for the QoS differentiated provisioning of services. As part of it, we have presented a flexible QoS preference model which improves the state-of-the-art in policy-based preference specification by combining cardinal and ordinal preferences. Combining attribute-value assertions with conditional attribute-value assertions and utility-value assertions provides policy authors greater flexibility and expressivity in specifying their preferences. Using our preference model, both domain experts and naive users can easily specify their preferences over the customizable QoS attributes with varying degrees of expressivity and complexity. The task of evaluating service requests and offers can be delegated to autonomous agents which can reason about the QoS preferences and take decisions on behalf of the service consumers and providers. We have used simple examples to illustrate the benefits of using the utility-value assertion and the conditional attribute-value assertions when expressing preferences over non-functional attributes. We have also shown how, given our QoS preference model, the QoS evaluation problem can be mapped to a CSP without any loss of semantics. The future plan includes building mechanisms to detect and resolve conflicting preferences and also introducing negotiation mechanisms to facilitate automated agreement establishment in case no solution exists which satisfies both consumer requirements and provider preferences.

ACKNOWLEDGMENT

This work was partially funded by the Service Aggregation Project within the Smart Services CRC which is proudly supported by the Australian Federal Government’s CRC Grant Program. It was also partially supported by the European Union in the scope of the European Regional Development Fund Program No.POIG.01.03.01-00-008/08.

REFERENCES


