User Preference Based Automated Selection of Web Service Compositions

Sudhir Agarwal and Steffen Lamparter

Institute of Applied Informatics and Formal Description Methods (AIFB),
University of Karlsruhe (TH), Germany.
{agarwal,lamparter}@aifb.uni-karlsruhe.de

Abstract. Semantically rich descriptions of web services enable automatic composition and matchmaking. End users as well as providers compose web services according to their needs. Since in general, there can be more than one possibilities (combinations of web services) to achieve a certain goal, a user has to decide which of the alternatives suits him the best. This leads to the requirement that the combinations of web services must be comparable. This can be achieved by aggregating web service attributes. Since the ranking of web service combinations depends on user's needs, capabilities and willingness for investment of resources, considering user preferences is important while calculating the rank of a web service composition. We show, how user preferences as well as aggregation information can be modeled in a formal way and how web service combinations can be ranked automatically based on this information.

1 Introduction

Composition of web services is performed by end users as well as web service providers. End users compose web services because (1) there may be no single web service that directly offers the desired functionality (2) a combination of web services may need less investment or capabilities than a single service. Providers compose web services in order to offer the composite service as a new web service. That is, the new composite web service acts as a mediator between an end user and other (component) web services.

However, there are in general more than one combinations of web services that can fulfill a given goal. That is, there is a need for decision support to help an end user to select one combination from many combinations. Such a decision support is only possible if the combinations are comparable. To compare two combinations of web services, the values of attributes of the component web services must be aggregated according to the structure of the composition. From the point of view of a provider aggregation is needed in order know how much he has to invest for the component web services so that he can decide how much he can demand for his composite service. In the following, we will not differentiate between end users and providers any further.

Consider a simple web service combination consisting of a sequence of web services w_1 and w_2 having price p_1 and p_2 respectively. Now, if a user wants to

know, how much the execution of the combination costs him, he needs to add p_1 and p_2. The values p_1 and p_2 are specified in the descriptions of the web services w_1 and w_2. Note, that in general, there is a need for machine support to perform such calculations, since (1) the number of web service combinations can be large, (2) the structure of the web service combinations can be complex, (3) a user may need to do such calculations very often. The most straightforward solution of our above problem would be, that the user inserts the descriptions of the web service combinations together with descriptions of the component web services and queries the knowledge base for the desired value, e.g. $p_1 + p_2$. This leads to the requirement, that the underlying user's description logic must support functions like addition, multiplication, minimum and maximum of numbers. In description logics theory, such functions are called aggregation functions and are first introduced in [1].

In this paper, our focus is on the selection phase. We make use of the techniques proposed in [1] to model aggregation information. In section 2, we will discuss some important non-functional properties of web services and show how they can be aggregated, when web services are composed. We show, how the aggregation information can be modeled as part of the ontology for describing web service combinations. In section 3, we show how user preferences can be modeled with fuzzy rules. In section 4, we will show how web services can be ranked based on the user preferences. We conclude in section 6 after discussing some related work in section 5.

2 Modeling Aggregation Information

While WSDL is commonly used to describe functional properties of a web service, there are currently no standards for describing non-functional properties of a web service and for describing composite web services. In general, every user can define his own ontologies to describe non-functional properties of his web service and to describe web service combinations. But, we believe, that in many cases users will use one of the already existing ontologies that are supported by their tools. Current ontologies for describing web service combinations do not allow to model the aggregation information about various attributes of component web services. Modeling aggregation information as part of a plan ontology is our main focus in this section. Having such aggregation information as part of description of a web service combination, values of various attributes of component web services can be aggregated automatically and a user only needs to specify his preferences to rank the plans.

2.1 Description of Non-Functional Properties

All those properties that are not absolutely necessary to be able to invoke a service and integrate the output are referred to as non-functional properties. In this section, we introduce the most common non-functional properties related to the direct usage of a web service. Note that only properties of the web service usage

itself and not of the product derived by means of a web service are discussed here.

Quality of Service. The most frequently discussed non-functional properties are quality of service attributes. These are attributes that define a minimal level of quality a service has to provide. They cover different aspects: The property *Locative Availability* defines a time frame a service is online, e.g. every day from 6 AM to 8 PM. Furthermore, an *Availability-Rate* can be defined that specifies the minimal percentage a service has to be available within a day, e.g. an Availability-Rate of 0.9 defines that a service has to be available at least 90% of a day. *Response Time* measures the duration between sending out the request and receiving a result from the service.

Price. A service price refers to the monetary amount that has to be paid by the requester to be allowed to use the service. Thus, the price is defined by an absolute *amount* and a specific *currency*. Furthermore, there may be *discounts* and *penalties* that can have influence on the price of a service.

Payment Method. To specify the price that has to be paid, properties like *Payment Instrument*, the *Duration*, and the *Charging Style* are required. Payment instrument refers to the type of payment, e.g. paying cash, by credit card, voucher, etc. The duration defines the time period in which the payment has to be completed. Moreover, services can be invoiced in different ways, which is specified by charging style (e.g. per invocation, for a certain time period).

Security. Security attributes allow to define *Identification Methods* as well as *Encryption Methods* that are supported by a service. Identification Methods refer to X509 [2], Kerberos [3], PIN input or similar systems. Encryption Methods defines the supported protocols for communication with a service.

Trust. There are various different ways to define trust levels with respect to web services. One way would be check for referrals by means of reputation systems. In this case a *Rating* attribute can be used to specify the level of trust within a certain integer range.

Privacy. Privacy statements basically describe what happens with the personal data of a requester once sent to a service. Such statements define, for instance, the allowed *Storage Period* of the data or if the service provider is allowed to disclose the data to third parties. This is expressed by the attribute *Disclosure*, which is valued by 'yes' or 'no'.

Of course, this list of non-functional properties is not exhaustive. A more detailed description of such properties can be found in [4].

2.2 Description of Web Service Combinations

Having discussed the non-functional properties an atomic web service, we now turn our attention to the description of web service combinations. We refer to such combinations as plans, processes or composite web services. We model four types of compositions, namely *sequence*, *parallel*, *choice* and *loop*. In this paper, our aim is to describe only static aspects of a composite web service. For doing so, we use standard description logic syntax and refer to [5] for details about the

semantics.

$$\texttt{Sequence} \sqsubseteq \texttt{WS} \sqcap \exists \texttt{component}.\texttt{WS}$$
$$\texttt{Parallel} \sqsubseteq \texttt{WS} \sqcap \exists \texttt{component}.\texttt{WS}$$
$$\texttt{Choice} \sqsubseteq \texttt{WS} \sqcap \exists \texttt{component}.\texttt{WS}$$
$$\texttt{Loop} \sqsubseteq \texttt{WS} \sqcap \exists \texttt{ws}.\texttt{WS} \sqcap \exists \texttt{times}.\mathbb{N}$$

Note, that we do not require to model the execution semantics of the various constructs in order to model aggregation information about the non-functional properties of the web service combinations. The roles `ws` and `times` are functional roles. That is, an instance of `Loop` can be related to only one instance of `WS` via the relation `ws` and to only one natural number via the relation `times`.

2.3 Modeling Aggregation Information

In section 2.1, we discussed some important non-functional properties of web services. While aggregating web service compositions only domain independent attributes are relevant.

	Sequence	Parallel	Choice	Loop
Locative Availability (LA)	*overlap*	*overlap*	*overlap*	LA of `ws`
Availability-Rate (AR)	$\prod$	min	min	*exp*((AR of `ws`), `times`)
Response Time (RT)	$\sum$	max	max	*mult*((RT of `ws`)), `times`)
Price Amount (PA)	$\sum$	$\sum$	max	*mult*((PA of `ws`), `times`)
Encryption Method (EM)	$\bigcup$	$\bigcup$	$\bigcup$	EM of `ws`
Identification Method (IM)	$\bigcup$	$\bigcup$	$\bigcup$	IM of `ws`
Rating (R)	min	min	min	R of `ws`
Storage Period (SP)	max	max	max	SP of `ws`
Disclosure (D)	$\bigvee$	$\bigvee$	$\bigvee$	D of `ws`

Fig. 1. Aggregation Functions

The function symbols used in figure 1 have their obvious meanings except the functions overlap, exp and mult. The function overlap determines the overlapping range of time periods $t_1, \ldots, t_n \in t$. The functions exp(x,y) and mult(x,y) calculate x^y and $x \times y$ respectively.

Now, we turn our attention to how the aggregation information as summarized figure 1 in can be modeled as part of the ontology for describing web service combinations that we described in section 2.2.

$$\texttt{Sequence} \sqsubseteq \prod_{i \in \{1,\ldots,n\}} P_{=}(a_i, \Sigma^S_{a_i}(\texttt{component} \circ a_i)) \tag{1}$$

where $\Sigma^S_{a_i}$ is an aggregation function for values of property a_i when composed in a sequence. E.g. a_i is *response time*, $\Sigma^S_{a_i}$ is equal to $\sum$.

$$\texttt{Parallel} \sqsubseteq \bigsqcap_{i \in \{1,\ldots,n\}} P_{=}(a_i, \Sigma^P_{a_i}(\texttt{component} \circ a_i)) \tag{2}$$

where $\Sigma^P_{a_i}$ is an aggregation function for values of property a_i when composed parallely. E.g. if a_i is *response time*, $\Sigma^P_{a_i}$ is equal to *max*.

$$\texttt{Choice} \sqsubseteq \bigsqcap_{i \in \{1,\ldots,n\}} P_{=}(a_i, \Sigma^C_{a_i}(\texttt{component} \circ a_i)) \tag{3}$$

where $\Sigma^C_{a_i}$ is an aggregation function for values of property a_i when composed as alternatives. E.g. if a_i is *response time*, $\Sigma^P_{a_i}$ is equal to *max*.

$$\texttt{Loop} \sqsubseteq \bigsqcap_{i \in \{1,\ldots,n\}} P_{=}(a_i, \Sigma^L_{a_i}(\texttt{ws} \circ a_i, \texttt{times})) \tag{4}$$

where $\Sigma^C_{a_i}$ is an aggregation function for values of property a_i when composed in a loop. E.g. if a_i is *response time*, $\Sigma^L_{a_i}$ is equal to *mult*.

3 User Preference Modeling

In many application domains, not only the membership of an individual to a set is nonrigid, but also the transition between the memberships of an individual from one set to another is smooth. Consider, for example, *height* of a human. Small children grow, but when do they stop to be small? So the transition from short humans to humans of average height is rather smooth and not crisp. Such kinds of knowledge can be encoded using techniques from fuzzy logic.

Vague knowledge, i.e. rules based on fuzzy logic, are also important from the perspective of evaluating values of attributes that have very complex dependencies with other attribute values. Such rules play an important role in application domains, where a good approximation of the desired value of an attribute is acceptable. For example, consider the controlling of a train. It is desired, that when a train arrives at station, it halts at a certain fixed position. However, calculating how exactly the brake should be applied at what position in which speed so that the passengers can still sit comfortable etc. is difficult. Considering that it is acceptable if the train stops a small distance before or after the mark, automatic control of the train is much easier.

Fuzzy logic, first introduced by Zadeh in [6, 7], provides answers to both the problems. On the one hand, the fuzzy sets allow to model vague memberships of individuals to sets. On the other hand, fuzzy IF-THEN rules allow to evaluate good approximations of desired attribute values in a very efficient way [6, 7].

In this section, we will show how user preferences can be modeled with fuzzy IF-THEN rules. We begin with the modeling of fuzzy membership functions and show how the membership of an individual to a fuzzy membership function can be calculated inside an appropriate description logic reasoner. Then, we model fuzzy IF-THEN rules and show how the degree of fulfillment of a rule by an individual can be calculated.

3.1 Modeling Fuzzy Membership Functions

Figure 2 shows the linguistic terms *fast*, *medium* and *slow* modeled as membership functions for a linguistic variable *Response Time*.

Now let $\mathbb{R}_{[0,1]}$ denote the set of real numbers between 0 and 1. For a concept v and a linguistic term t, we define a membership function μ_t^v as a finite and non-empty set of points[1] (x, y) in $\mathbb{R} \times \mathbb{R}_{[0,1]}$, where x is a number representing[2] an individual of the concept v. We will define next a concept *Point*. For this purpose, we introduce two concrete functional roles x and y, which assign the first respectively second coordinate to the point. So we define the concept *Point* as $Point \sqsubseteq \exists x.\mathbb{R} \sqcap \exists y.\mathbb{R}_{[0,1]}$.

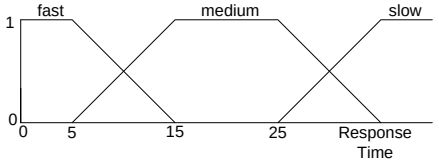

Fig. 2. Example Membership Functions

Similarly, we define a concept μ that denotes the set of membership functions. We do this by means of a (non-functional) role p which assigns points to individuals, as $\mu \sqsubseteq \exists p.Point$[3].

For a concept v, being viewed as a linguistic variable and having linguistic terms $t_1, \ldots, t_n$, we add n instances $\mu_{t_1}^v, \ldots, \mu_{t_n}^v$ of μ with corresponding roles and points. We interpret the set of points associated with some $\mu_{t_i}^v$ as a piecewise linear function. For a concept v being viewed as a linguistic variable, we denote the set of its linguistic terms by v^*.

We refer to [8] for complete modeling of the information needed to calculate the membership of a given individual to a given fuzzy set. As an example, the response time of 12 time units is medium with degree 0.7 and fast with degree 0.3 considering the fuzzy sets from figure 2.

3.2 Modeling Fuzzy Rules

A *fuzzy IF-THEN rule* consists of an IF part (antecedent) and a THEN part (consequent). The antecedent is a combination of terms, whereas the consequent is exactly one term. In the antecedent, the terms can be combined by using fuzzy conjunction, disjunction and negation. A *term* is an expression of the form $X = T$, where X is a linguistic variable and T is one of its linguistic terms.

Since terms are the elementary building blocks of a fuzzy rule, we start with modeling terms. As described above, a term consists of two parts, a linguistic variable and a linguistic term. So, we model a concept *Term* as $Term \sqsubseteq \exists r.\top \sqcap \exists f.\mu$, where the roles r and m are functional roles assigning linguistic

[1] I.e. we allow only membership functions which are piecewise linear.

[2] Identifying individuals with real numbers simply serves to make computations simpler, though it may appear to be counterintuitive in some cases.

[3] $\mu \sqsubseteq \geq 2\, p.Point$ would be more precise if qualified number restrictions are available.

variable resp. linguistic term. Terms can be combined via conjunction, disjunction and negation to term expressions. Further, a term expression is fulfilled by an individual to a certain degree. So, we define concept *TermExp* and extend the definition of the concept *Term* as follows:

$$\begin{aligned} \mathit{TermExp} &\sqsubseteq \exists \mathit{degree}.\mathbb{R}_{[0,1]} \\ \mathit{Term} &\sqsubseteq \mathit{TermExp} \end{aligned}$$

A rule has an antecedent and a consequent. The antecedent is a term expression and the consequent is a term. Further, a rule has a degree to which it is fulfilled by an individual. So, we define a concept *Rule* as[4]

$$\mathit{Rule} \sqsubseteq \exists \mathit{antecedent}.\mathit{TermExp} \sqcap \exists \mathit{consequent}.\mathit{Term} \sqcap \exists \mathit{degree}.\mathbb{R}_{[0,1]}.$$

3.3 Calculating the Degree of Fulfillment of a Rule

Since terms are the basic building blocks of a rule, the degree of fulfillment of a rule depends ultimately on the degrees of fulfillment of the terms occurring in the rule. Now, an individual connected to a term via the role r fulfills the term with the same degree as the corresponding value of the membership function the term is connected with via the role m. We model this by extending the concept *Term* with the axiom $\mathit{Term} \sqsubseteq P_{=}(\mathit{degree}, r \circ m_f)$.

We can calculate the degree of fulfillment of a term expression according to the semantics suggested by Zadeh in [6, 7], which can be summarized as follows.[5] Given two membership functions μ_A and μ_B, $(\mu_A \wedge \mu_B)(a) = \min\{\mu_A(a), \mu_B(a)\}$, $(\mu_A \vee \mu_B)(a) = \max\{\mu_A(a), \mu_B(a)\}$ and $(\neg\mu_A)(a) = 1 - \mu_A(a)$.

So, we define the concept $\mathit{TermExp}_{\wedge}$, $\mathit{TermExp}_{\vee}$ and $\mathit{TermExp}_{\neg}$ as follows, introducing also the corresponding roles[6].

$$\begin{aligned} \mathit{TermExp}_{\wedge} &\sqsubseteq \mathit{TermExp} \sqcap \exists \mathit{conjunct}.\mathit{TermExp} \sqcap P_{=}(\mathit{degree}, \min\{\mathit{conjunct} \circ \mathit{degree}\}) \\ \mathit{TermExp}_{\vee} &\sqsubseteq \mathit{TermExp} \sqcap \exists \mathit{disjunct}.\mathit{TermExp} \sqcap P_{=}(\mathit{degree}, \max\{\mathit{disjunct} \circ \mathit{degree}\}) \\ \mathit{TermExp}_{\neg} &\sqsubseteq \mathit{TermExp} \sqcap \exists \mathit{operand}.\mathit{TermExp} \sqcap P_{=1-}(\mathit{degree}, \mathit{operand} \circ \mathit{degree}) \end{aligned}$$

The predicate $P_{=1-}(a, b)$ is true iff $a = 1 - b$. min and max are aggregate functions for the concrete domain $\mathbb{R}$.

To interpret a fuzzy IF-THEN rule, we need an interpretation for the implication. In general, one can have a different interpretation of the implication for every rule, which is particularly important when the application domain requires the use of weighted rules. Here, we use a universal interpretation π of the implication in all the rules. However, we do not fix π any further. In most of the cases, it is equal to minimum. So,

$$\mathit{Rule} \sqsubseteq P_{\pi}(\mathit{degree}, \mathit{antecedent} \circ \mathit{degree}, \mathit{consequent} \circ \mathit{degree}),$$

[4] The roles *antecedent* and *consequent* are functional roles.
[5] Certainly, other T-norms and T-conorms could be used.
[6] The relation *operand* is a function role.

where P_π is a ternary predicate from the concrete domain $\mathbb{R}$ and represents the interpretation of the implication function π. That is, for given $a, b, c \in \mathbb{R}$, $P_\pi(a, b, c)$ is true iff $a = \pi(b, c)$.

3.4 Modeling User's Preference as Fuzzy Rules

We view preferences as the information that describes the constraints on the properties of an individual in order to be accepted for further consideration. We specify different levels of acceptance with fuzzy membership functions as described above.

In the web service compositions scenario, the individuals are concrete web service compositions. We model user preferences with fuzzy IF-THEN rules. The IF part contains membership functions of the various properties of an individual (e.g. those listed in figure 1) and the THEN part is one of the membership functions of a special concept called *Rank*. Intuitively, a fuzzy rule describes which combination of attribute values a user is willing to accept to which degree, where attribute values and degree of acceptance are fuzzy sets, i.e. vague. Note, that a user has to define at most as many rules as there are degrees of acceptance that he/she wants to differentiate. We believe, that in practice, the number of such categories will not be large. An example fuzzy IF-THEN rule can be

```
IF RT = fast and PA = cheap THEN Rank = high.
```

4 Automated Plan Selection

Let o represent the concept that represents the acceptance and let o be categorized in k categories represented by $g_1 \dots g_k$. Further, there exists k rules $R_1, \dots, R_i, \dots, R_k$, where R_i has g_i as conclusion. In the following, we calculate the ranking r of an individual a with respect to objective o according to the FITA principle (First Inferencing Then Aggregation). The other alternative for interpreting fuzzy rules is FATI (First Aggregation Then Inferencing). It has been shown in [9] that the two principles are equivalent.

4.1 FITA

Consider a rule base containing n rules of the form $F_1 \rightarrow G_1, \dots F_n \rightarrow G_n$. In FITA, first each rule is interpreted. That is, for each x, y and each rule i, the value of $\pi(F_i(x), G_i(y))$ is calculated. Now, for a given x, the inference step is performed for each rule. Again, the inference operator can be different for different rules. However, we use a universal inference operator for all the rules and call it κ — in many practical cases, κ is equal to minimum. In general, the inference operator κ is some function that maps the square $[0, 1]^2$ to $[0, 1]$. Performing an inference step for a given x and a given rule i means calculating $\kappa(F(x), \pi(F_i(x), G_i(y)))$, where F is some fuzzy set describing the membership function for a given situation. Having performed the inferencing step for all the

n rules, an aggregation step is performed to obtain a single value from n values. For this purpose, an aggregation operator $\alpha : [0,1]^n \rightarrow [0,1]$ is needed. The most common aggregation operator is maximum. However, we do not fix α any further. In the aggregation step,

$$\alpha(\kappa(F(x), \pi(F_1(x), G_1(y))), \ldots \kappa(F(x), \pi(F_n(x), G_n(y))))$$

is calculated. We define a concept *FITA* as:

$$\begin{aligned} \mathit{FITA} \sqsubseteq & \prod_{i\in\{1,\ldots,n\}} \exists \mathit{rule}_i.\mathit{Rule} \sqcap \prod_{i\in\{1,\ldots,n\}} \exists x_i.\mathbb{R}_{[0,1]} \sqcap \exists \mathit{output}.\mathbb{R}_{[0,1]} \sqcap \\ & \prod_{i\in\{1,\ldots,n\}} P_\kappa(x_i, \mathit{rule}_i \circ \mathit{degree}, x \circ m_f) \sqcap P_=(\mathit{output}, \alpha\{x_i\}) \end{aligned}$$

where α is an aggregate function and P_κ is a ternary predicate on the concrete domain $\mathbb{R}_{[0,1]}$. $P_\kappa(a,b,c)$ is true iff $a = \kappa(b,c)$.

4.2 Defuzzification

The goal of a DL query is to determine the value of an instance. FITA delivers the membership of an arbitrary instance of the target concept to the goal fuzzy concept according to the compositional rule of inference. That is, if we have a sufficient number of instances of the target concept, we can calculate for each instance its membership to the goal fuzzy concept. This way, we obtain a set of points $(x, \mu_T(x))$, where x is an arbitrary instance of the target concept and μ_T is the target fuzzy concept.

However, the goal of query answering is to determine an instance of the target concept. This is done by interpreting the set of points $(x, \mu_T(x))$ as an area in $\mathbb{R}^2$ and defuzzifying this area. One of the most common defuzzification methods is the so called *center of gravity* method, where the geometrical center of gravity of a given area is calculated. The desired instance is then equal to the value of the x-coordinate of the center of gravity of the area. Hence, the desired instance ω can be calculated by the following formula:

$$\omega = \frac{\int x \cdot \mu(x) dx}{\int \mu(x) dx} \tag{5}$$

To model the defuzzification process, we define a concept *DefuzInfo* as: $\mathit{DefuzInfo} \sqsubseteq \exists \mathit{fita}.\mathit{FITA} \sqcap \exists x \sqcap \exists \mathit{prod}.\mathbb{R} \sqcap P_{mul}(\mathit{prod}, x, \mathit{fita} \circ \mathit{output})$.

Finally, consider a concept $C \sqsubseteq \exists w.W$. For a given instance θ of C, to determine an instance ω such that $w(\theta, \omega)$ holds while considering fuzzy rules, we extend the definition of the concept C as follows:

$$C \sqsubseteq \exists \mathit{di}.\mathit{DefuzInfo} \sqcap P_=(w, x) \sqcap P_{div}(x, \mathit{sum}\{\mathit{di} \circ \mathit{prod}\}, \mathit{sum}\{\mathit{di} \circ \mathit{fita} \circ \mathit{output}\}). \tag{6}$$

We use the already existing instances of the concept W as the arbitrary instances for determining the area that is defuzzified. If no instances of W are

available in the knowledge base, we can always insert some instances which do not need to be in any relation with instances of other concepts. The number and value of such instances depends on the application domain, more precisely on the width of the range (subset of $\mathbb{R}$) and on the value of dx in equation 5.

4.3 Calculation of Ranking of Web Service Compositions

In the above sections, we have explained how an attribute value of an individual that has complex dependencies on other attribute values of the same individual can be calculated automatically, where the dependencies are modeled as fuzzy rules. Coming back to our big picture described in section 1, we have the situation that a user has many web service compositions, each of them fulfilling user's goal, and the user needs to select one of them for execution. That is, we have many web service compositions and we wish to calculate rank for each of them based on user's preferences.

We calculate the rank of a web service composition by setting C equal to WS and w equal to $rank$ in equation 6. Further, we set range of $rank$ equals to $Rank$. That is, $WS \sqsubseteq \exists rank.Rank$. The rank of a web service composition is then the value of the attribute $rank$. Finally, we perform this step for each web service composition. Since, we have already modeled the calculation of aggregation of various attribute values in equations 1, 2, 3 and 4, they are automatically considered in the calculation of the overall ranking.

4.4 Example

Consider two web services w_1 and w_2. Suppose both the web services w_1 and w_2 use the web services c_1, c_2 and c_3 with the only difference that w_1 uses them in sequence whereas w_2 calls them in parallel. Suppose the response times of c_1, c_2 and c_3 are 3, 4 and 5 time units resp. Suppose, a user is only concerned with the response time of the web services and has the following preferences modeled as fuzzy rules

```
  IF RT = fast THEN Rank = high
IF RT = medium THEN Rank = average
  IF RT = slow THEN Rank = low
```

The response times rt_1 and rt_2 of w_1 and w_2 are calculated with equations 1 and 2 as 12 and 5 time units resp. Degrees of fulfilments of the rules by w_1 are 0.3, 0.7 and 0 resp. and by w_2 1.0, 0 and 0 resp. (cf. figure 2). Considering, "low", "medium" and "high" as fuzzy sets for the concept "Rank", step FITA yields one area for each web service. For w_1 this area is the maximum of the areas for the sets low, medium and high chopped at 0, 0.7 and 0.3 and for w_2 the maximum of of the areas chopped at 0, 0 and 1.0. Now, the center of gravities g_1 and g_2 of both the areas are calculated in the defuzzification step. g_1 lies left to g_2 which means w_1 is ranked lower than w_2, which corresponds to the intuition.

5 Related Work

There are quite a few efforts for modeling web processes. OWL-S [10] and BPEL4WS [11] are the most widely known. OWL-S claims to have formal semantics and thus added value as compared to XML based approaches like BPEL4WS. However, none of them allows to model aggregation information, which is necessary to reason about composite web services. We believe, that OWL-S and similar ontologies can become more useful by using our approach to enable formal specification of aggregation information and thus allowing reasoning about properties of web service combinations and making them comparable.

There are a few approaches that have investigated how various attributes of workflows or composite web services can be aggregated [12–14]. Our approach builds on the existing works, since it uses the insights gained from the mentioned works. The mentioned works does not provide a mechanism how the aggregation information can be modeled as part of a process ontology. We have shown, that ranking and thus plan selection is a possible use case for aggregation of web service attributes. We have also shown how rankings can be calculated and plan selection can be automated. Most of the existing approaches for automatic selection e.g. [15] either consider only atomic services or they are not based on user preferences.

6 Conclusion and Outlook

In this paper, we presented a generic approach for modeling aggregation information for various web service attributes as part of an ontology for describing composite web services. We have also shown, how user preferences can be modeled as fuzzy rules. Consequently, with the techniques presented in this paper, web service combinations can be compared with each other and ranked according to the user preferences.

In our previous works [16], we have developed a tool for automatic composition of web services. Currently, a user has to go through all the generated plans and decide manually, which plan he wishes to execute. We intend to extend this tool by a plan selection component based on the approach presented in this paper. The tool will allow a user to enter his preferences and present the user a list of generated plans sorted by rank. On the basis of which the user can decide more easily which of the plans, he wishes to execute. Further, a user can define a priori a threshold for rank of a plan, which he wishes to be executed automatically.

In this paper, we suggested to select composite services based on a ranking calculated with respect to user preferences. However, this "take it or leave it" principle may be sometimes economically inefficient, since possible negotiations yielding higher benefits for both sides are not considered [17]. In future, we wish to investigate the possibility of integrating multi attributive negotiations that facilitates Pareto-optimal allocations into our system.

Acknowledements

This work was funded by the Federal Ministry of Education and Research (BMBF), the German Research Foundation (DFG), and the European Union in scope of the Internetökonomie project SESAM, the Graduate School Information Management and Market Engineering and the IST project SEKT under contract IST-2003-506826.

References

1. Baader, F., Sattler, U.: Description Logics with Concrete Domains and Aggregation. In Prade, H., ed.: Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98), John Wiley & Sons Ltd (1998) 336–340
2. OpenSSL: Openssl x509. (http://www.openssl.org/docs/apps/x509.html)
3. MIT: Kerberos. (http://web.mit.edu/kerberos/www/)
4. O'Sullivan, J., Edmond, D., ter Hofstede, A.: Formal description of non-functional service properties. Technical report, Queensland University of Technology (2005)
5. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F., eds.: The Description Logic Handbook: Theory Implemenation and Applications. Cambridge University Press (2003)
6. Zadeh, L.A.: Fuzzy Sets. Information and Control **8** (1965) 338–353
7. Yager, R.R., Ovchinnikov, S., Tong, R.M., Nguyen, H.T., eds.: Fuzzy sets and applications - Selected Papers by L. A. Zadeh. John Wiley & Sons, New York, NY, USA (1987)
8. Agarwal, S., Hitzler, P.: Modeling Fuzzy Rules with Description Logics. In: Proceedings of Workshop on OWL Experiences and Directions, Galway, Ireland (2005)
9. Temme, K.H., Thiele, H.: On the correctness of the principles of FATI and FITA and their equivalence. In: IFSA 95 - Sixth International Fuzzy Systems Association World Congress. Volume II. (1995) 475–478
10. Coalition, D.S.: DAML-S: Web Service Description for the Semantic Web. In: ISWC2002: Ist International Semantic Web Conference, Sardinia, Italy. LNCS, Springer (2002) 348–363
11. et al., T.A.: Business Process Execution Language for Web Services. Technical report, Bea Systems, IBM Corp., Microsoft Corp., SAP AG, Siebel Systems (2003)
12. Cardoso, J., Sheth, A., Miller, J., Arnold, J., Kochut, K.: Quality of Service for Workflows and Web Service Processes. Journal of Web Semantics **1** (2004)
13. Cardoso, J., Sheth, A.: Semantic e-Workflow Composition. Journal of Intelligent Information Systems **21** (2003) 191–225
14. Jaeger, M.C., Rojec-Goldmann, G., Mühl, G.: QoS Aggregation in Web Service Compositions. In: Proceedings of IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE-05), China, IEEE Press (2005) 181–185
15. Liu, Y., Ngu, A.H., Zeng, L.Z.: Qos computation and policing in dynamic web service selection. In: WWW Alt. '04: Proc. of 13th Int. WWW Conf. on Alternate track papers & posters. (2004) 66–73
16. Agarwal, S., Handschuh, S., Staab, S.: Annotation, Composition and Invocation of Semantic Web Services. Journal of Web Semantics **2** (2005) 1–24
17. Bichler, M.: An experimental analysis of multi-attribute auctions. Decision Support Systems **29** (2000)