Report October 12th

1 Formal definition of the Rhone service based query rewriting algorithm

The basic input for the Rhone algorithm is: (1) a query; (2) a list of concrete services.
Definition 1 (Query): A query Q is defined as a set of *abstract services*, a set of *constraints*, and a set of *user preferences* in accordance with the grammar:

$$Q(\overline{I}, \overline{O}) := A_1(\overline{I}, \overline{O}), A_2(\overline{I}, \overline{O}), .., A_n(\overline{I}, \overline{O}), C_1, C_2, .., C_m[P_1, P_2, .., P_k]$$

The left side of the definition is called the *head* of the query; and the right side is called the *body*. $\overline{I}$ and $\overline{O}$ are a set of *input* and *output* parameters, respectively. Input parameters in both sides of the definition are called *head variables*. In contrast, input parameters only in the query body are called *local variables*. Abstract services $(A_1, A_2, .., A_n)$ describes a set of basic service capabilities. $C_1, C_2, .., C_m$ are constraints over the *input* and/or *output* parameters. The user preferences (over the services) are signed in $P_1, P_2, .., P_k$. C and P are in the form $x \otimes constant$ such that $\otimes \in \{\geq, \leq, =, \neq, <, >\}$.

To illustrate the definition, let us suppose the set of abstract services in Table 1 and the Example 1.

Abstract Service	***Description***
DiseaseInfectedPatients(d?,p!)	Given a disease *d*, a list of patients *p* infected by it is retrieved.
PatientDNA(p?,dna!)	Given a patient *p*, his DNA information *dna* is retrieved.
PatientPersonalInformation(p?,info!)	Given a patient *p*, his personal information *info* is retrieved.

Table 1: Abstract services description

Example 1: *The user wants to retrieve the DNA information from patients infected by the disease 'K' using services that have availability higher than 99%, price per call less than 0.2 dollars, and the total cost less then 1 dollar.*

The query which express the Example 1 according to the Definition 1 and the abstract services in Table 1 is specified below. The decorations ? and ! are used to specify input and output parameters, respectively.

$$Q(d?, dna!) := DiseaseInfectedPatients(d?, p!), PatientDNA(p?, dna!),$$
$$d = \text{“}K\text{”}[availability > 99\%,\ price\ per\ call < 0.2\$,\ total\ cost < 1\$]$$

Analyzing the query, it is possible to note that the parameters “d?” and “dna!” appear in both sides of the definition. Due to that they are *head* variables. On the other hand, “p!” and “p?” are *local* variables considering that they appear only in the body definition. Additionally, note that the local variables “p!” and “p?” have the same name. Intuitively, this fact indicates a dependency between the abstract services which use these variables (in that case *DiseaseInfectedPatients* and *PatientDNA*).

In the example, *DiseaseInfectedPatients* and *PatientDNA* are abstract services that specify basic service functions which are combined to answer the query. The constraint ($d =$ “K”) over the input parameter 'd' will be further used while executing the query over a database

(the where clause). *Availability*, *price per call* and *total cost* are the user preferences over the services.

Definition 2 (Concrete service): A concrete service (S):

$$S(\overline{I}, \overline{O}) := A_1(\overline{I}, \overline{O}), A_2(\overline{I}, \overline{O}), .., A_n(\overline{I}, \overline{O})[P_1, P_2, .., P_k]$$

A concrete service (S) is defined as a set of abstract services (A), and by its quality constraints P. These quality constraints associated to the service represent the service level agreement exported by the concrete service.

Example 2: Considering the query (see Example 1) and the abstract services (see Table 1), the concrete services below are examples in accordance with the Definition 2.

$S1(a?, b!) := DiseaseInfectedPatients(a?, b!)[availability > 99\%,\ price\ per\ call = 0.2\$]$
$S2(a?, b!) := DiseaseInfectedPatients(a?, b!)[availability > 99\%,\ price\ per\ call = 0.1\$]$
$S3(a?, b!, c!) := DiseaseInfectedPatients(a?, b!, c!)[availability > 98\%,\ price\ per\ call = 0.1\$]$
$S4(a?, b!) := PatientDNA(a?, b!)[availability > 99.5\%,\ price\ per\ call = 0.1\$]$
$S5(a?, b!) := PatientDNA(a?, b!)[availability > 99.7\%,\ price\ per\ call = 0.1\$]$
$S6(a?, b!) := PatientPersonalInformation(a?, b!)[availability > 99.7\%,\ price\ per\ call = 0.1\$]$
$S7(a?, b!) := PatientDNA(a?, c!), PatientPersonalInformation(c?, b!)[availability > 99.7\%,\ price\ per\ call = 0.1\$]$

Given the query and a list of concrete services as input, the algorithm looks for candidate concrete services. Candidate concrete service is a concrete service that probably can be used in the rewriting process. It contains only abstract services which are also query abstract services, and with the same signature (same name and number of input/output variables). The candidate concrete services are chosen while searching for matches between abstract services in S and abstract service in Q.

Definition 3 (abstract service equivalence): A match between abstract services occurs when an abstract service A_i is equivalent to A_j, denoted $A_i = A_j$. Given two abstract services A_i and A_j, $A_i = A_j$ iff: (1) A_i and A_j have the same abstract function name; (2) the number of *input* parameters of A_i is equal to A_j; and (3) the number of *output* parameters of A_i is equal to A_j. For example, looking to the concrete services in the Example 2, the abstract service *DiseaseInfectedPatients* in $S1$ and $S2$ are equivalent to the abstract service *DiseaseInfectedPatients* in the query Q (Example 1) once they have the same name and number of input/output parameters. On the other hand, the abstract service *DiseaseInfectedPatients* in $S3$ is not equivalent to the abstract service *DiseaseInfectedPatients* in the query because the number of parameters are different.

Based on the assumptions that: (a) a concrete service can represent a service composition in which the abstract services involved may be able not only to retrieve data, but also to execute business rules that may impact the entire system; and (b) the execution of a concrete service consists in executing all its abstract services. A concrete service (S) is selected as *candidate* to the rewriting process if for each abstract service in S there is an equivalent in Q; there is no abstract service in S that does not exist in Q; and the quality constrains in Q must be guaranteed in S.

Definition 4 (candidate service): Given a query Q and a concrete service S, S is a *candidate* service iff: (1) $\nexists A_i\ s.t.\ A_i \in S\ and\ A_i \notin Q$; and (2) the quality constraints in S does not violate the user preferences in Q.

For example, considering the query in the Example 1 and the concrete services in the Example 2, it is possible to see that: (1) $S1$ is not a candidate service because it violates an user preference (*price per call*); (2) $S3$ and $S7$ are not a candidate service because they have abstract services that are not in Q; and (3) $S2$, $S4$ and $S5$ are candidate services once: all their abstract services have an equivalent in Q and there is no violation in the user preference.

A *candidate service description* (CSD) describes how a *candidate* concrete service can be used in the query rewriting process. It is a complex data structure which includes: mappings from variables in a concrete service to variables in the query; mappings from variables on the head of a concrete service to variables on its body; a set of abstract services that represents partially or fully the abstract services in the query; and a set of quality constrains associated to the concrete service. Intuitively, a rewriting is a set of *candidate service descriptions* that fully covers the original query, and do not violates the user preferences.

Definition 5 (candidate service description): A CSD is represented by an n-tuple:

$$\langle S, h, \varphi, G, P \rangle$$

where S is a concrete service. h are mappings between variables in the head of S to variables in the body of S. φ are mapping between variables in the concrete service to variables in the query. G is a set of abstract services covered by S. P is a set quality constraints associated to the service S.

The CSD for a given service will be created following rules: (1) for all head variables in S, there is a mapping for a head variable in Q; and (2) if x is an local variable in S mapped to a local variable in Q, then S must cover all abstract services in Q which uses x or cover only one abstract service that uses x.

Example 3: To illustrate the rules above consider the following example. *The user wants to retrieve the personal information and the DNA information from patients infected by disease "K"*. Supposing we have the query Q and the concrete services $S1$, $S2$, $S3$ and $S4$:

$$Q(d?, info!, dna!) :=$$
$$DiseaseInfectedPatients(d?, p!), PatientDNA(p?, dna!), PatientPersonalInformation(p?, info!)$$
$$S1(a?, b!) := DiseaseInfectedPatients(a?, c!), PatientDNA(c?, b!)$$
$$S2(a?, b!) := PatientPersonalInformation(a?, b!)$$
$$S3(a?, b!) := DiseaseInfectedPatients(a?, b!)$$
$$S4(a?, b!) := PatientDNA(a?, b!)$$

In the query Q it is possible to note that "p!" is a *local* variable which is used as input ("p?") for the abstract services $A2$ and $A3$. Looking to the concrete service $S1$ no CSD will be created for it because the *local* variable $c!$ is mapped to the local variable $p!$, but $S1$ does not cover all abstract services which expects that variable. On the other hand, CSDs are constructed to the services $S2$, $S3$ and $S4$ once even existing the mapping from a local variable in the concrete service to a local variable in the query, all of them only cover one abstract service which uses that *local* variable. To be more clear about these rules, consider the rewriting below in which the CSDs for the services $S2$, $S3$ and $S4$ are used:

$$Q(d?, info!, dna!) := S3(d?, p!), S4(p?, dna!), S2(p?, info!)$$

The rewriting above is the only one possible for the query. However, let us suppose that a CSD for $S1$ was created violating the rule number two, consequently the wrong rewriting below would be created:

$$Q(d?, info!, dna!) := S1(d?, info!), S4(p?, dna!)$$

The problem here is regarding the *local* variable $p?$ which appears in $S4$, and it apparently should come from $S1$, but we can not guarantee that the same *local* variable internally used in $S1$ is the one expected by $S4$. That is the reason the rule two exists.

2 Query rewriting approaches

A query rewriting algorithm which processes queries on data provider services is proposed in [1]. A query is defined using SPARQL and data services are modeled as RDF views. Both can be seen as a graph. Given a query and a set of data services, the algorithm searches for relevant services and creates a mapping table for them. This table shows different ways of using a data service to cover part of the query. Then, based on the mapping table, the algorithm generates different combinations of data services to answer the query. A valid combination (a rewriting answer) is a service composition in which the set of data service graphs fully satisfy the query graph.

A service composition framework to answer preference queries is proposed in [2]. The concept of preferences is included to SPARQL queries, and fuzzy constraints to services. Services and service compositions are ranked according to a fuzzy dominance relationship and fuzzy scores. Two algorithms based on [1] to generate the rewriting compositions are presented: (1) the first produces all possible rewritings before computing their scores, and then return the best ones; (2) the second uses a quality metric that combines diversity and accuracy to, incrementally, rank services and to build the best rewritings.

References

[1] M. Barhamgi, D. Benslimane, and B. Medjahed. A query rewriting approach for web service composition. *Services Computing, IEEE Transactions on*, 3(3):206–222, July 2010.

[2] Karim Benouaret, Djamal Benslimane, Allel Hadjali, and Mahmoud Barhamgi. FuDoCS: A Web Service Composition System Based on Fuzzy Dominance for Preference Query Answering, September 2011. VLDB - 37th International Conference on Very Large Data Bases - Demo Paper.