# Query Rewriting Algorithm for Data Integration Quality

Daniel A. S. Carvalho
Univ. Lyon 3 - Lyon, France
first.last@univ-lyon3.fr

Plácido A. Souza Neto
IFRN - Natal, Brazil
first.last@ifrn.edu.br

Chirine Ghedira-Guegan
Univ. Lyon 3 - Lyon, France
first.last@univ-lyon3.fr

Nadia Bennani
CNRS-INSA - Lyon, France
first.last@insa-lyon.fr

Genoveva Vargas-Solar
CNRS-LIG - Grenoble, France
first.last@imag.fr

## ABSTRACT

This paper introduces the general lines of a rewritring algorithm named *Rhone* that addresses query rewriting for data integration. The originality of *Rhone* is that the rewriting process is guided by quality measures associated to data providers (services) and user preferences. The paper uses a running scenario to describe the *Rhone*'s implementation and gives some hints about its experimental evaluation.

## 1. INTRODUCTION

Data integration has evolved with the emergence of data services that deliver data under different quality conditions related to data freshness, cost, reliability, availability, among others. Data are produced continuously and on demand in huge quantities and sometimes with few associated metadata, which makes the integration process more challenging. Some approaches express data integration as a service composition problem where given a query the objective is to lookup and compose data services that can contribute to produce a result. Finding the best service composition that can answer a query can be computationally costly. Furthermore, executing the composition can lead to retrieve and process data collections that can require important memory, storage and computing resources. This problem has been addressed in the service-oriented domain [2, 3, 1]. Generally, these solutions deal with query rewriting problems. [2] proposed a query rewriting approach which processes queries on data provider services. [3] introduced a service composition framework to answer preference queries. In that approach, two algorithms based on [2] are presented to rank the best rewritings based on previously computed scores.

[1] presented an algorithm that produces and order rewritings according to user preferences. Yet, to our knowledge few works consider quality measures associated both to data services and to user preferences in order to guide the rewriting process.

This paper introduces the early stages of our ongoing work on developing the *Rhone* service-based query rewriting algorithm guided by SLA's. Our work addresses this issue and proposes the algorithm *Rhone* with two original aspects: (i) the user can express her quality preferences and associate them to her queries; and (ii) service's quality aspects defined on Service Level Agreements (SLA) guide service selection and the whole rewriting process.

Section 2 describes the algorithm *Rhone*, proposed in our work. Section 3 describes a running scenario and also implementation issues. Finally, section 4 concludes the paper.



## 2. SERVICE-BASED QUERY REWRITING ALGORITHM

This section describes *Rhone* the service-based query rewriting algorithm that we propose. Given a set of *abstract services*, a set of *concrete services*, a *user query* and a set of user *quality preferences*, derive a set of service compositions that answer the query and that fulfill the quality preferences.

The input for Rhone is: (1) a query; (2) a list of concrete services defined in the following lines.

**Definition 1 (Query):** A query $Q$ is defined as a set of *abstract services* ($A$), a set of *constraints* ($C$), and a set of *user preferences*($P$) 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*. The preferences (for short *measures*) are classified as: *single measures* (static measures) and *composed measures* (defined as aggregations of *single measures*).

**Definition 2 (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 (SLA) exported by the concrete service.

The algorithm consists in four steps: (i) select candidate concrete services; (ii) create mappings from concrete services to the query (called *concrete service description (CSD)*); (iii) combine the list of CSDs; and finally (iv) produce rewritings from the query $Q$.

**Select candidate concrete services:** This step consists in looking for concrete services that can be matched with

**Algorithm 1** - RHONE

```
1: function rhone(Q, S)
2:   L_S ← SelectCandidateServices(Q, S)
3:   L_CSD ← CreateCSDs(Q, L_S)
4:   I ← CombineCSDs(Q, L_CSD)
5:   R ← ∅
6:   p ← I.next()
7:   while p ≠ ∅ and T_init⟦ Agg(Q) ⟧ do
8:     if isRewriting(Q, p) then
9:       R ← R ∪ Rewriting(p)
10:        T_inc⟦ Agg(Q) ⟧
11:    end if
12:    p ← I.Next()
13:  end while
14:  return R
15: end function
```

the query (line 2). In this sense, there are three matching problems: (i) *abstract service matching*, an abstract service $A$ can be matched with an abstract service $B$ only if $(a)$ they have the same name, and $(b)$ they have a compatible number of variables; (ii) *measure matching*, all *single measures* in the query must exist in the concrete service, and all of them can not violate the measures in the query ; and (iii) *concrete service matching*, a concrete service can be matched with the query if all its abstract services satisfy the *abstract service matching* problem and all the *single measures* satisfy the *measures matching* problem.

The result of this step is a list of *candidate concrete services* which may be used in the rewriting process.

**Creating concrete service descriptions:** In this step the algorithm tries to create *concrete services description* (CSD) to be used in the rewriting process (line 3). A CSD maps abstract services and variables of a concrete service to abstract services and variables of the query. A CSD is created according to variable mapping rules mainly based on 2 criterias: the type and the dependency (variables used as inputs on other abstract services). The result of this step is a list of CSDs.

**Combining CSDs.** Given all produced CSDs (line 4), they are combined among each other to generate a list of lists of CSDs, each element representing a possible composition.

**Producing rewritings.** The final step (lines 5-13) identifies which lists of CSDs are a valid rewritings of the user query given the list of lists of CSDs. A combination of CSDs is a valid rewriting if: (i) they cover all abstract services in the query; and (ii) there is mapping to all head variables in the query (implemented by the function $isRewriting(Q,p)$ - line 8). The originality of our algorithm concerns the aggregation function ($\mathcal{A}gg(Q)$). It is responsible to check and increment *composed measures* (if present in the query). This means for each element in the CSD list the value of *composed measure* is incremented (line 10), and rewritings are produced while the values of these measures are respected (line 7). The result of this step is the list of valid rewritings of the query (line 14).

# 3. IMPLEMENTATION AND RESULTS

Let us suppose the following medical scenario to illustrate our service-based query rewriting algorithm. Users can access *abstract services* (basic service capabilities) to retrieve: (i) patients infected by a given disease (*DiseasePatients(d?,p!)*); (ii) patient dna information *PatientDNA(p?, dna!)*; and (iii) patient personal information (*PatientInformation(p?, info!)*). The decorations ? and ! are used to specify input and output parameters, respectively.

Let us consider the query: *a user wants to retrieve personal and DNA information of patients who were infected by a disease 'K' using services that have availability higher than 98%, price per call less than 0.2 dollars, and total cost less than 1 dollar.* The query below corresponds to the example (see Definition 1).

$$Q(d?, dna!) := DiseasePatients(d?, p!), PatientDNA(p?, dna!), \\ [availability > 99\%,\ price\ per\ call < 0.2\$,\ total\ cost < 1\$]$$

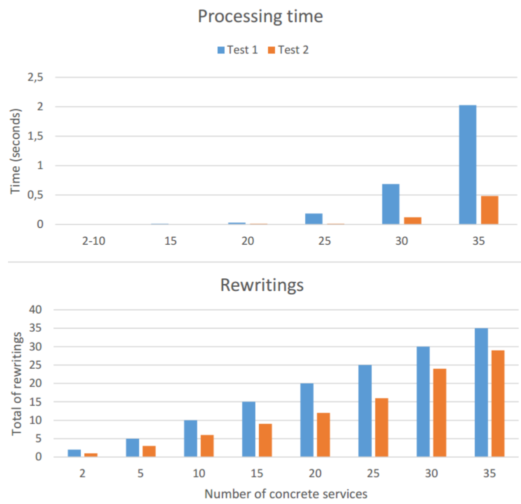


Figure 1: Query rewriting evaluation.

We use 7 concrete services to run our approach. In this example all the queries have 6 *abstract services* and 2 *single measures*. The number of local variables (dependencies) and CSDs is being modified to see how the algorithm works under these conditions.

By now, the analysis identified that the factor that influenciates the Rhone performance is the number of CSDs versus the number of abstract services in the query since they increase the number of possible combinations of CSDs. We proceeded two types of analysis for the query $Q$ (*Test 1* and *Test 2* - Figure 1). The first set of tests doesn't consider quality measures, while the second consider the measures in the rewriting process. The number of concrete services used in the analysis is from 2 until 35 services. Our preliminary results show that our algorithm, considering the quality measures, presents a better result for performance and the total number of rewritings.

# 4. CONCLUSIONS

This work proposes a query rewriting algorithm for data integration quality named *Rhone*. Given a query and a list of concrete services as input, the algorithm looks for candidate concrete services. These candidate services can be used probably in the rewriting process to match the query. We are currently performing experiments in order to better evaluate the performance of the *Rhone*.

# 5. REFERENCES


[1] C. Ba, U. Costa, M. Halfeld Ferrari, R. Ferre, M. A. Musicante, V. Peralta, and S. Robert. Preference-Driven Refinement of Service Compositions. In *CLOSER (4th International Conference on Cloud Computing and Services Science)*, Proceedings of CLOSER 2014, page 8 pages, Barcelona, Spain, Apr. 2014. POSTER presentation.

[2] 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.

[3] K. Benouaret, D. Benslimane, A. Hadjali, and M. Barhamgi. FuDoCS: A Web Service Composition System Based on Fuzzy Dominance for Preference Query Answering, Sept. 2011. VLDB - 37th International Conference on Very Large Data Bases - Demo Paper.

[4] A. Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, Dec. 2001.