# Query Rewriting Algorithm for Data Integration Quality

Daniel A. S. Carvalho
Magellan, IAE, Univ-Lyon3, France
first.last@univ-lyon3.fr

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

Chirine Ghedira-Guegan
LIRIS, IAE, Univ-Lyon3, France
first.last@univ-lyon3.fr

Nadia Bennani
LIRIS, 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.

The remainder of this paper is organized as follows. 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 and discusses our work perspectives.



## 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 regarding the context of data service deployment.

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* (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 ($A_i$), and by their quality constraints $P_j$. 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.

**Algorithm 1** - RHONE

1: **function** $rhone(Q, \mathcal{S})$
2: $\mathcal{L}_{\mathcal{S}} \leftarrow SelectCandidateServices(Q, \mathcal{S})$
3: $\mathcal{L}_{\mathcal{CSD}} \leftarrow CreateCSDs(Q, \mathcal{L}_{\mathcal{S}})$
4: $I \leftarrow CombineCSDs(\mathcal{L}_{\mathcal{CSD}})$
5: $R \leftarrow \emptyset$
6: $p \leftarrow I.next()$
7: **while** $p \neq \emptyset$ **and** $\mathcal{T}_{\text{init}}[\![\ \mathcal{A}gg(Q)\ ]\!]$ **do**
8: **if** $isRewriting(Q, p)$ **then**
9: $R \leftarrow R \cup Rewriting(p)$
10: $\mathcal{T}_{\text{inc}}[\![\ \mathcal{A}gg(Q)\ ]\!]$
11: **end if**
12: $p \leftarrow I.Next()$
13: **end while**
14: **return** $R$
15: **end function**

**Select candidate concrete services:** This step looks for concrete services that can be matched with 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:** This step 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 onto 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

The Rhone prototype is implemented in Java. There are 15 classes in which 14 of them model the basic concepts (*query*, *abstract services*, *concrete services*, etc) and 1 responsible to implement the core of the algorithm.

Currently, our approach runs in a controlled environment. Two experiments were produced: *Test 1* and *Test 2*. In both the query contains 6 *abstract services* and 4 quality *measures*. The service registry used has 35 concrete services.

In each experiment, there are a set of tests in which the number of concrete services varies from 2 to 35. The important difference between the experiments is regarding the use of quality preferences to guide the service selection and query rewriting. *Test 1* do not consider quality measures. On the other hand, *Test 2* consider them. The figure 1 shows our first results.

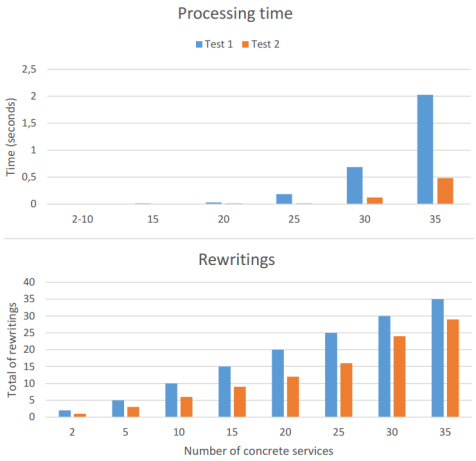


Figure 1: Query rewriting evaluation.

The analysis identified that the algorithm shares the same problem as existing query rewriting approaches using views while it does not consider quality measures (*Test 1*): increasing the processing time when the query and the number of concrete services increase. However, while considering the quality measures (*Test 2*), the results are promising. The *Rhone* increases performance reducing rewriting number which allows to go straightforward to the rewriting solutions that are satisfactory avoiding any further backtrack and thus reducing successful integration time.

## 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. H. Ferrari, R. Ferre, M. A. Musicante, V. Peralta, and S. Robert. Preference-Driven Refinement of Service Compositions. In *Int. Conf. on Cloud Computing and Services Science, 2014*.

[2] M. Barhamgi, D. Benslimane, and B. Medjahed. A query rewriting approach for web service composition. *IEEE Transactions on Services Computing*, 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. In *Int. Conf. on Very Large Data Bases, 2011*.