**DESCRIPTION OF THE ALGORITHMS**

1. **Algorithm I**

The algorithm I deals with queries that are equivalents in terms of *abstract services*. This means that the set of *abstract services* of an incoming query is equal to the set of *abstract services* of a query in the query history. The steps below describes the algorithm I:

***Step 1*:** Search in the history for previous queries that have exactly the same abstract services included in the incoming query.

***Step 2*:** For the set of queries retrieved in the *step 1*, select the queries that have requirements equivalent or more restrict than the incoming query. If there is result for this step, go to *step 4*. Otherwise, continue in the *step 3*.

***Step 3*:** For the set of queries retrieved in the *step 1*, which consequently have requirements less restrict than the incoming query, order them according to their timestamp, type (Processed or Reused) and score giving priority to the most recent, processed and higher scored queries. Thus, pick the first one and use it on the *step 7b*.

***Step 4*:** For the set of queries retrieved in the *step 2*, select the queries that have requirements equivalent. If there is result for this step, go to *step 5*. Otherwise, go to *step 6*.

***Step 5*:** For the set of queries retrieved in the *step 4*, order them according to their timestamp and type (Processed or Reused) giving priority to the most recent and processed queries. Thus, pick the first one and use it on the *step 7*.

***Step 6*:** For the set of queries retrieved in the *step 2*, order them according to their timestamp, type (Processed or Reused) and score giving priority to the most recent, processed and higher scored queries. Thus, pick the first one and use it on the *step 7*.

***Step 7*:** Given a query to be reused, identify the compositions produced to this query that have online services and order them according to their score. Pick the first composition, return it to be executed, and go to *step 8*.

***Step 7b*:** Given a query to be reused, identify the compositions produced to this query that have online services, which satisfy the requirements, and order them according to their score. Pick the first composition, return it to be executed, and go to *step 8*. [Otherwise, pick the higher scored query, even without covering completely the requirement, return it to be executed if the user agrees according the percentage of requirements that this query covers. I think here could appear the degree of satisfaction that we talked in the last meeting.]

***Step 8*:** Save the new query in the history as “reused” query, copying the compositions selected in the *step 7* and the services used by these compositions.

1. **Algorithm II**

The algorithm II deals with queries that are superset in terms of *abstract services*. This means that the set of *abstract services* of an incoming query is contained in the set of *abstract services* of a query in the query history. The steps below describes the algorithm II:

***Step 1*:** Search in the history for previous queries that have a set of abstract services bigger than the incoming query, but containing the abstract services included in the incoming query.

***Step 2*:** For the set of queries retrieved in the *step 1*, select the queries that have requirements equivalent or more restrict than the incoming query. If there is result for this step, go to *step 4*. Otherwise, continue in the *step 3*.

***Step 3*:** For the set of queries retrieved in the *step 1*, which consequently have requirements less restrict than the incoming query, order them according to their timestamp, type (Processed or Reused) and score giving priority to the most recent, processed and higher scored queries. Thus, pick the first one and use it on the *step 7b*.

***Step 4*:** For the set of queries retrieved in the *step 2*, select the queries that have requirements equivalent. If there is result for this step, go to *step 5*. Otherwise, go to *step 6*.

***Step 5*:** For the set of queries retrieved in the *step 4*, order them according to their timestamp and type (Processed or Reused) giving priority to the most recent and processed queries. Thus, pick the first one and use it on the *step 7*.

***Step 6*:** For the set of queries retrieved in the *step 2*, order them according to their timestamp, type (Processed or Reused) and score giving priority to the most recent, processed and higher scored queries. Thus, pick the first one and use it on the *step 7*.

***Step 7*:** Given a query to be reused, identify the compositions produced to this query that have as online services the ones that covers that abstract services of the incoming query, and order them according to their score. Pick the first composition and go to *step 8*.

***Step 7b*:** Given a query to be reused, identify the compositions produced to this query that have as online services the ones that covers that abstract services of the incoming query, which satisfy the requirements, and order them according to their score. Pick the first composition and go to *step 8*. Otherwise, pick the higher scored query, even without covering completely the requirement and go to *step 8b*.

***Step 8*:** Given a composition, (i) make its projection including only the services that covers the abstract services in the incoming query; (ii) validate the rewriting; (iii) if the rewriting is valid, return it to be executed and go to *step 9*. Otherwise, pick another composition in the *step 7* (or *step 7b* depending on which *step* called *step 8*) and perform again the *step 8*.

***Step 8b*:** Given a composition, (i) make its projection including only the services that covers the abstract services in the incoming query; (ii) validate the rewriting; (iii) if the rewriting is valid, return it to be executed if the user agrees according to the percentage of requirements that this query covers and go to *step 9*.

***Step 9*:** Save the new query in the history as “reused” query, copying the projection of the compositions selected in the *step 7* that are a valid rewrite of the query and the services used by these compositions.

1. **Algorithm III**

The algorithm III deals with queries that are subset in terms of *abstract services*. This means that the set of *abstract services* of an incoming query contains the set of *abstract services* of a query in the query history. The steps below describes the algorithm III:

***Step 1*:** Search in the history for previous queries that have a set of abstract services included in the set of abstract services of the incoming query.

***Step 2*:** For the set of queries retrieved in the *step 1*, select the queries that have requirements equivalent or more restrict than the incoming query. If there is result for this step, go to *step 4*. Otherwise, continue in the *step 3*.

***Step 3*:** For the set of queries retrieved in the *step 1*, which consequently have requirements less restrict than the incoming query, order them according to their timestamp, type (Processed or Reused), score and size (number of abstract services) giving priority to the most recent, processed, higher scored, and bigger queries. Thus, pick the first one and use it on the *step 7b*.

***Step 4*:** For the set of queries retrieved in the *step 2*, select the queries that have requirements equivalent. If there is result for this step, go to *step 5*. Otherwise, go to *step 6*.

***Step 5*:** For the set of queries retrieved in the *step 4*, order them according to their timestamp, type (Processed or Reused), and size (number of abstract services) giving priority to the most recent, processed, and bigger queries. Thus, pick the first one and use it on the *step 7*.

***Step 6*:** For the set of queries retrieved in the *step 2*, order them according to their timestamp, type (Processed or Reused) and score giving priority to the most recent, processed and higher scored queries. Thus, pick the first one and use it on the *step 7*.

***Step 7*:** Given a query *Q1* selected in the *step 5*, search for a query *Q2* that complements *Q1* using the query selection process in the algorithm I. Select the compositions of *Q1* and *Q2* that have online services, which satisfy the requirements (depending on the type of the requirements *Q2*), and order them according to their scores. Pick the first composition of each query and go to *step 8*.

***Step 7b*:** Given a query *Q1* selected in the *step 5*, search for a query *Q2* that complements *Q1* using the query selection process in the algorithm I. Select the compositions of *Q1* and *Q2* that have online services, which satisfy the requirements and order them according to their scores. Pick the first composition of each query and go to *step 8*. Otherwise, pick the higher scored query, even without covering completely the requirement and go to *step 8b*.

***Step 8*:** Given two compositions, (i) combine them; (ii) validate the rewriting; (iii) if the rewriting is valid, return it to be executed and go to *step 9*. Otherwise, pick another composition in the *step 7* (or *step 7b* depending on which *step* called *step 8*) and perform again the *step 8*.

***Step 8b*:** Given two compositions, (i) combine them; (ii) validate the rewriting; (iii) if the rewriting is valid, return it to be executed if the user agrees according to the percentage of requirements that this query covers and go to *step 9*.

***Step 9*:** Save (i) the new query in the history as “reused” query, (ii) the valid rewritings produced by combing the queries selected in the *step 7* and (iii) the services used by these compositions.

**NEW ALGORITHM**

**Input:** an incoming query Qin, which includes a set of abstract services (A1..An) and a set of user integration requirements (R1..Rm).

**Output:** the best reusable service composition in accordance with the incoming query.

1: **function** *reuseQueries* (Qin)

2: LQ ← Ø

3: LQ ← *searchForQueries* (Qin, “equivalent”)

4: **if** LQ ≠ Ø **then**

5: *p* ← LQ.*next* ()

6: LRC ← Ø

7: **while** *p* ≠ *null* **do**

8: LRC ← LRC U *p.getOnlineCompositions* ()

9:  *p* ← LQ.*next* ()

10: **end while**

11: LRC ← *validateCompositions* (LRC)

12: *storeResults* (Qin, LRC)

13: **return** *first* (LRC)

14: **end if**

15:LQ ← *searchForQueries* (Qin, “superset”)

16: **if** LQ ≠ Ø **then**

17: *p* ← LQ.*next* ()

18: LRC ← Ø

19: **while** *p* ≠ *null* **do**

20: LRC ← LRC U *p.getOnlineCompositions* ()

21: LRC ← *projectCompositions* (LRC)

22:  *p* ← LQ.*next* ()

23: **end while**

24: LRC ← *validateCompositions* (LRC)

25: *storeResults* (Qin, LRC)

26: **return** *first* (LRC)

27: **end if**

28:LQ ← *searchForQueries* (Qin, “subset”)

29: **if** LQ ≠ Ø **then**

30: *p* ← LQ.*next* ()

31: LRC ← Ø

32: **while** *p* ≠ *null* **do**

33:LRC’ ← Ø

34: LCQ ← Ø

35: LRC’ ← LRC’ U *p.getOnlineCompositions* ()

36: LCQ ← *searchForComplementaryQuery* (*p*)

37: *p’* ← LCQ.*next* ()

38: **while** *p’* ≠ Ø **do**

39:LRC’’ ← Ø

40: LRC’’ ← *p’.getOnlineCompositions* ()

41:LRC ← LRC U *combineCompositions* (LRC’, LRC’’)

42: *p’* ← LCQ.*next* ()

43: **end while**

44:  *p* ← LQ.*next* ()

45: **end while**

46: LRC ← *validateCompositions* (LRC)

47: *storeResults* (Qin, LRC)

48: **return** *first* (LRC)

49: **end if**

50: **end function**

The idea behind this algorithm is given an incoming query Qin, *(i)* to find a set of previous stored queries that could be reused in order to answer the user request, *(ii)* to identify all available compositions associated to these previous queries, and *(iii)* to select the best composition for answering the query.

*First*, the algorithm searches for previous queries that could be used to answer the incoming query (line 2). The method *searchForQueries* (Qin) executes a stored procedure that retrieves the reusable queries according to their type. For example, *equivalent* queries have priority over *superset* queries, and *superset* queries have priority over *subset* queries. Thus, the set of queries retrieved belongs to single type respecting this priority. *Second*, the algorithm iterates (lines 5 to 22) over the set of queries (LQ) performing the necessary operations to reuse them. If LQ includes *equivalent* queries or *superset* queries, *(i)* the algorithm filters the available compositions associated to each query in this set, and *(ii)* it projects the part of the composition that can answer the user request. Otherwise, the algorithm searches for queries that can be complementary (*searchForComplementaryQuery*, line 12) and iterates on them (lines 14 to 19) combining the available compositions (lines 16 and 17). *Finally*, the reusable compositions are validated (*validateCompositions*, line 23), the results stored for a next integration request (*storeResults*, line 24), and the best composition, which satisfies the user requirements, is returned (line 25).