NEW ALGORITHM

Input: an incoming query Qin, which includes a set of abstract services ($A_1..A_n$) and a set of user integration requirements ($R_1..R_m$).

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

1: **function** *reuseQueries* (Q_{in})
2: $L_Q \leftarrow$ *searchForQueries* (Q_{in})
3: $p \leftarrow L_Q$.*next* ()
4: $L_{RC} \leftarrow \emptyset$
5: **while** $p \neq \emptyset$ **do**
6: **if** *p.isEquivalent* (Q_{in}) **or** *p.isSuperset* (Q_{in}) **then**
7: $L_{RC} \leftarrow L_{RC} \cup$ *p.getOnlineCompositions* ()
8: $L_{RC} \leftarrow$ *projectCompositions* (L_{RC})
9: **else**
10: $L_{RC'}, L_{CQ} \leftarrow \emptyset$
11: $L_{RC'} \leftarrow L_{RC'} \cup$ *p.getOnlineCompositions* ()
12: $L_{CQ} \leftarrow L_{CQ} \cup$ *searchForComplementaryQuery* (*p*)
13: *p'* $\leftarrow L_{CQ}$.*next* ()
14: **while** *p'* $\neq \emptyset$ **do**
15: $L_{RC''} \leftarrow \emptyset$
16: $L_{RC''} \leftarrow L_{RC''} \cup$ *p'.getOnlineCompositions* ()
17: $L_{RC} \leftarrow L_{RC} \cup$ *combineCompositions* ($L_{RC'}, L_{RC''}$)
18: *p'* $\leftarrow L_{CQ}$.*next* ()
19: **end while**
20: **end if**
21: $p \leftarrow L_Q$.*next* ()
22: **end while**
23: $L_{RC} \leftarrow$ *validateCompositions* (L_{RC})
24: *storeResults* (Q_{in}, L_{RC})
25: **return** *first* (L_{RC})
26: **end function**

The idea behind this algorithm is given an incoming query Q_{in}, *(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* (Q_{in}) 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 (L_Q) performing the necessary operations to reuse them. If L_Q 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).