Report 26 April

What are the *derived SLA* and *integration SLA*? How do you structure them to reuse and optimize a further integration? Where can we find them?

The *derived SLA* is a type of SLA that includes information about the query, user preferences, user cloud subscriptions, the different data services and data processing services that can be used to answer the query, the compositions generated using data services, the compositions that were executed while meeting the user preferences and subscriptions, and where the results are delivered. The *derived SLA* is build dynamically during the whole integration process. When the results are delivered to the user, fulfilling his/her preferences and in accordance with his/her cloud subscriptions, an *integration SLA* is created based on the *derived SLA*. It will include the query, user preferences, user cloud subscriptions, the data services and data processing services that were used to answer the query, the compositions that were executed while meeting the user preferences and subscriptions, and where the results are delivered. The *integration SLA* is a sub-part of a *derived SLA*. *Integration SLA* and *derived SLA* are stored in a SLA registry in order to be reused in a further integration.

How does the whole process work?

Given a query, user preferences and specific user cloud subscriptions, the integration steps are:
Building the *derived SLA* including the query, user preferences and user cloud subscriptions.
Looking for previous *integration SLA*. We will search on our SLA registry for previous *integration SLA* that can be matched with the actual user query, preferences and cloud subscriptions. If a complete match is found, the information about used data services, generated and executed compositions are reused. If a partial match is found, for example, we have found an *integration SLA* with the same query, but with different preferences and cloud subscription restrictions, the data services that were listed as possible match with the query are reused.
Looking for data services. If no match can be found in the previous *integration SLA*s, we will proceed selecting data services that can be matched with the query. Here, we should use and extend the *Rhone*'s service selection step. Actually, the *Rhone* select services based on their definition, a structural match of service name and types of variables. I believe we should extend this process allowing a semantic match of services and variables. I think it should be done using ontologies. The result of this step is a set of services that can answer the query concepts in totality or partially.
Filtering data services. Data services selected in the previous step are filtered based on the user preferences and user cloud subscriptions. Here we also need to extend the *Rhone* filtering process. *Rhone* does not consider the restrictions imposed by the user cloud subscriptions, it just take into account the user preferences.

We have discussed in the last meeting that data services would be filtered considering the *SLAs* deduced from user's *SLAc*. However, reflecting today, I was not able to see how to automatically generate *SLAs* from different user's *SLAc*. The first reason is the *SLAs* should describe quality conditions the data service is delivered. Then, I believe this kind of information should be specified by who is providing the service (for example, the cloud that hosts the service, the owner of the service). Another reason is that the manipulation and combination of these *SLAs* could be not efficient considering that for each user cloud subscription a *SLAs* should be produced for each data service selected in the previous phase. This means in the worst case, if we have 10 services and 3 cloud subscriptions, we will produce 30 *SLAs*. I really do not know if it is interesting or not.

Concerning this issue, I believe that the cloud provider should define and furnish the different *SLAs* associated to each service. Thus, the filtering process is done matching the user preferences and user cloud subscriptions with the quality condition that a service is delivered specified in the *SLAs*.
Producing rewritings (service compositions). Given only the services that were filtered in the previous step, they are combined and checked if they are a rewriting of the query or not. Here we will use the another part of the *Rhone* for combining and producing rewritings. However, we need to extend it to consider the user cloud subscription while verifying and validating a rewriting.
Executing the integration. The best data processing service to execute the composition is selected considering the cost for retrieving, integrating and transferring data. The compositions are executed while the preferences and subscription restrictions are being satisfied.

Deliverables

- Examples of the different contracts (*derived, integration, SLAs and SLAc*)
- Matching example considering the SLAs
- Approach for matching the *derived SLA* with the *integration SLA*
- Extending *Rhone* selection process to include the semantic matching of services
- Algorithm for matching the user preferences and cloud subscription with the *SLAs* of the different services while selecting and rewriting the query (another extension to *Rhone*)

- Algorithm to evaluate the best location to execute the query considering the user cloud subscription and query economic cost
- Improve and detail the general process