Context Description and Next Activities

The figure 1 illustrates our data integration scenario. Cloud providers (for instance, Cloud Provider A, Cloud Provider B and Cloud Provider C) offer cloud resources to data providers (for instance, Data provider 1, Data provider 2, Data provider 3 and Data provider 4) willing to deploy their services. The cloud provider and the data service establish a contract specifying what guarantees in terms of infrastructure resources (for instance, storage limit, memory limit, processing capacity) the data service can expect from the cloud. This contract is called *Cloud SLA* (SLA_C).

Data services can deploy services in the clouds they have subscriptions respecting what is agreed in the SLA_C. Each service deployed by the data service in the cloud export a different SLA (called *Service SLA* - SLA_S) which specifies what service customers can expect in terms of data quality guarantees (for instance, provenance, freshness, data type, degree of rawness) from its service. The data provider defines the SLA_S for the services deployed on a cloud according to what it is defined in the SLA_C. For instance, considering that a *data provider* have agreed with a *cloud provider* to have limit of 10 gigabytes of free data transferred per day, the *data provider* could define on his SLA_S that he can perform 300 requests per day, and each request costs 0.1$. In other words, the SLA_S guarantees are derived from the SLA_C. Moreover, a *data provider* can deploy the same service in different clouds in which he has established contracts (for instance, the *Data provider* 1 deployed the service S1 in the clouds A and B) and for each different *Cloud provider* a different SLA_S is defined for the same service.

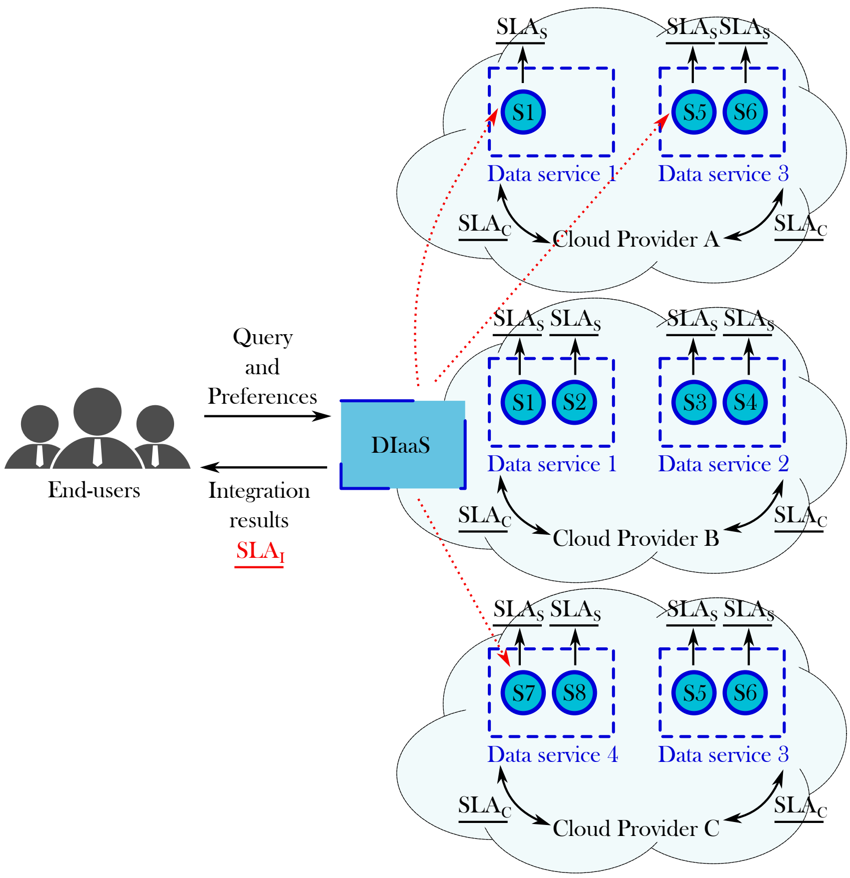

Figure 1: Data integration context overview

The end-user willing to integrate data interacts with our *Data Integration-as-a-Service* (*DIaaS*). The *DIaaS* is responsible to select and match the services that can produce the result expected by the user according to his preferences, where he is consuming the data, and the different SLAs associated to the services and to the cloud providers. Once the composition is created and executed, the integration results are delivered to the user and an *integration SLA* (SLA_I) is established. This SLA is responsible to include information collected during the integration process which can be reused in a further integration request.

Activities for the next meeting:

1. Enumerate the possible cases of queries and their variations to be able to identify what could be of interest to be in the *integration SLA* and what information should I collect in the integration process.

2. Propose the schemas of SLA_C and SLA_S according to the schema that I have removing what is not necessary for it (clauses extraction from SLA and how those clauses will feed the rewriting algorithm: study all the cases: clauses are compatibles, clauses are partially compatible...etc).

Definition 1 (*Abstract service*). An *abstract service* describes the small piece of function performed by a *service* deployed by a *data provider*. For instance, retrieve weather information, book a hotel, retrieve infected patients, among others. The *abstract service* is defined as follows: A ($\overline{I}$; $\overline{O}$) where A is the name which identifies the *abstract service*. $\overline{I}$ and $\overline{O}$ are a set of comma-separated input and output parameters, respectively.

Definition 2 (*Query*). An user *query* Q is defined as a sequence of *abstract services* followed by a set of *user requirements* in accordance with the grammar:

$$Q(\overline{I}_h; \overline{O}_h) := A_1(\overline{I}_{1l}; \overline{O}_{1l}), A_2(\overline{I}_{2l}; \overline{O}_{2l}), .., A_n(\overline{I}_{nl}; \overline{O}_{nl}), R_1, R_2, .., R_m$$

The *query* is defined in terms of *abstract services* $(A_1, A_2, .., A_n)$ including a set of *user requirements* $(R_1, R_2, .., R_m)$. The left-hand of the definition is called the *head*; and the right-hand is the *body*. A *query* Q includes a set of input $\overline{I}$ and output $\overline{O}$ variables, respectively. Variables in the *head* are identified by $\overline{I}_h$ and $\overline{O}_h$, and called *head* variables. They appear in the *head* and in the *body* definition. Variables appearing only in the *body* are identified by $\overline{I}_l$ and $\overline{O}_l$, and are called *local* variables. *Head* variables can be accessed and shared among different services. On the other hand, *local* variables can be used only by the service which define them.

Definition 3 (*User requirement*). An *user requirement* r is in the form $x \otimes c$, where x is an identifier; c is a constant; and $\otimes \in \{\geq, \leq, =, \neq, <, >\}$. The *user requirements* can be associated to single *service* or to the entire composition. For instance, the *total response time* is obtained by adding the *response time* of each *service* involved in the composition.

Definition 4 (*Requirement domain*). A *requirement domain (D)* groups *user requirements* which specify information in the same domain, and which can be comparable among each other. For instance, a *requirement domain time* includes time-related *user requirements* such as *response time*, *total response time*, *latency*, among others.

Definition 5 (*Requirement domain equivalence*). A *requirement domain* D_1 is equivalent to a *requirement domain* D_2, represented by $D_1 \cong D_2$, iff: $\forall r_i \in D_1,\ \exists r_j \in D_2 \mid r_i.x = r_j.x \textit{ and } |D_1| = |D_2|$.

Definition 6 (*Requirement domain containment*). A *requirement domain* D_1 is contained in a *requirement domain* D_2, represented by $D_1 \subset D_2$, iff: $\forall r_i \in D_1,\ \exists r_j \in D_2 \mid r_i.x = r_j.x \textit{ and } |D_1| < |D_2|$.

Definition 7 (*User requirements equivalence*). A set of *user requirements* R_1 is equivalent to a set of *user requirements* R_2, represented by $R_1 \cong R_2$, iff: $\forall r_i \in R_1,\ \exists r_j \in R_2 \mid eval(r_i) = eval(r_j) \textit{ and } |R_1| = |R_2|$.

Definition 8 (*User requirements more restrict*). Given a set of *user requirements* R_1 and R_2, R_1 is more restrict than R_2, represented by $R_1 \ \rhd\ R_2$, iff:

Case 1: $\forall r_i \in R_1,\ \exists r_j \in R_2 \mid eval(r_i) \subset eval(r_j) \textit{ and } |R_1| = |R_2|$.

Case 2: $\forall D_m \in R_1,\ \exists D_n \in R_2 \mid D_n \subset D_m$.

Definition 9 (*User requirements less restrict*). Given a set of *user requirements* R_1 and R_2, R_1 is less restrict than R_2, represented by $R_1 \ \lhd\ R_2$, iff:

Case 1: $\forall r_i \in R_1,\ \exists r_j \in R_2 \mid eval(r_i) \supset eval(r_j)\ and\ |R_1| = |R_2|$.

Case 2: $\forall D_m \in R_1,\ \exists D_n \in R_2 \mid D_m \subset D_n$.

Definition 10 (*Part of the user requirements more restrict and part less restrict*). Given a set of *user requirements* R_1 and R_2, part of the *user requirements* R_1 can be more restrict and part less restrict than the *user requirements* R_2, represented by $R_1\ \diamond\ R_2$, iff:

$\forall r_i \in R_1,\ \exists r_j \in R_2 \mid (eval(r_i) \subset eval(r_j)\ or\ eval(r_i) \supset eval(r_j))\ and\ |R_1| = |R_2|$.

Definition 11 (*User requirements different*). Given a set of *user requirements* R_1 and R_2, R_1 is different of R_2, represented by $R_1\ \neq\ R_2$, iff:

$\forall r_i \in R_1,\ \nexists r_j \in R_2 \mid r_i.x = r_j.x$.

Definition 7 (*Query equivalence*). Definition 2 (query equivalence): a query Q1 is equivalent to a query Q2 if: Q1 and Q2 have the same number of abstract services For each abstract services in Q1 there is an equivalent in Q2

Definition 8 (*Query subset*). Definition 10 (query subset): a query Q1 is a subset of the query Q2 if: Q1 has less abstract services than Q2 For each abstract service in Q1 there is an equivalent in Q2. In other words, the query Q1 is contained in the query Q2.

Definition 9 (*different queries but with some abstract services in common*). Definition 11 (different queries but with some abstract services in common): this case occurs when given a query Q1 and a query Q2: Q1 and Q2 have a different number of abstract services There is at least one abstract service in Q1 that has an equivalent in Q2; There is at least one abstract service in Q1 that has no equivalent in Q2; and There is at least one abstract service in Q2 that has no equivalent in Q1;

List of query variations

Query	Requirements
The incoming query is the same as a previous query	Same requirements
	Requirements more restrict
	Requirements less restrict
	Part of the requirements more restrict and part of the requirements less restrict
	Part of the requirements more restrict and part of the requirements different
	Part of the requirements less restrict and part of the requirements different
	Requirements completely different
The incoming query is a subset of a previous query	Same requirements
	Requirements more restrict
	Requirements less restrict
	Part of the requirements more restrict and part of the requirements less restrict
	Part of the requirements more restrict and part of the requirements different
	Part of the requirements less restrict and part of the requirements different
	Requirements completely different
The previous query is a subset of the incoming query	Same requirements
	Requirements more restrict
	Requirements less restrict
	Part of the requirements more restrict and part of the requirements less restrict
	Part of the requirements more restrict and part of the requirements different
	Part of the requirements less restrict and part of the requirements different
	Requirements completely different
Different queries but the incoming query has some abstract services in common with the previous query	Same requirements
	Requirements more restrict
	Requirements less restrict
	Part of the requirements more restrict and part of the requirements less restrict
	Part of the requirements more restrict and part of the requirements different
	Part of the requirements less restrict and part of the requirements different
	Requirements completely different
Different queries	Same requirements
	Requirements more restrict
	Requirements less restrict
	Part of the requirements more restrict and part of the requirements less restrict

Query	Requirements
	Part of the requirements more restrict and part of the requirements different
	Part of the requirements less restrict and part of the requirements different
	Requirements completely different

Table 1: List of possible query variations

Cloud SLA (Figure 2)

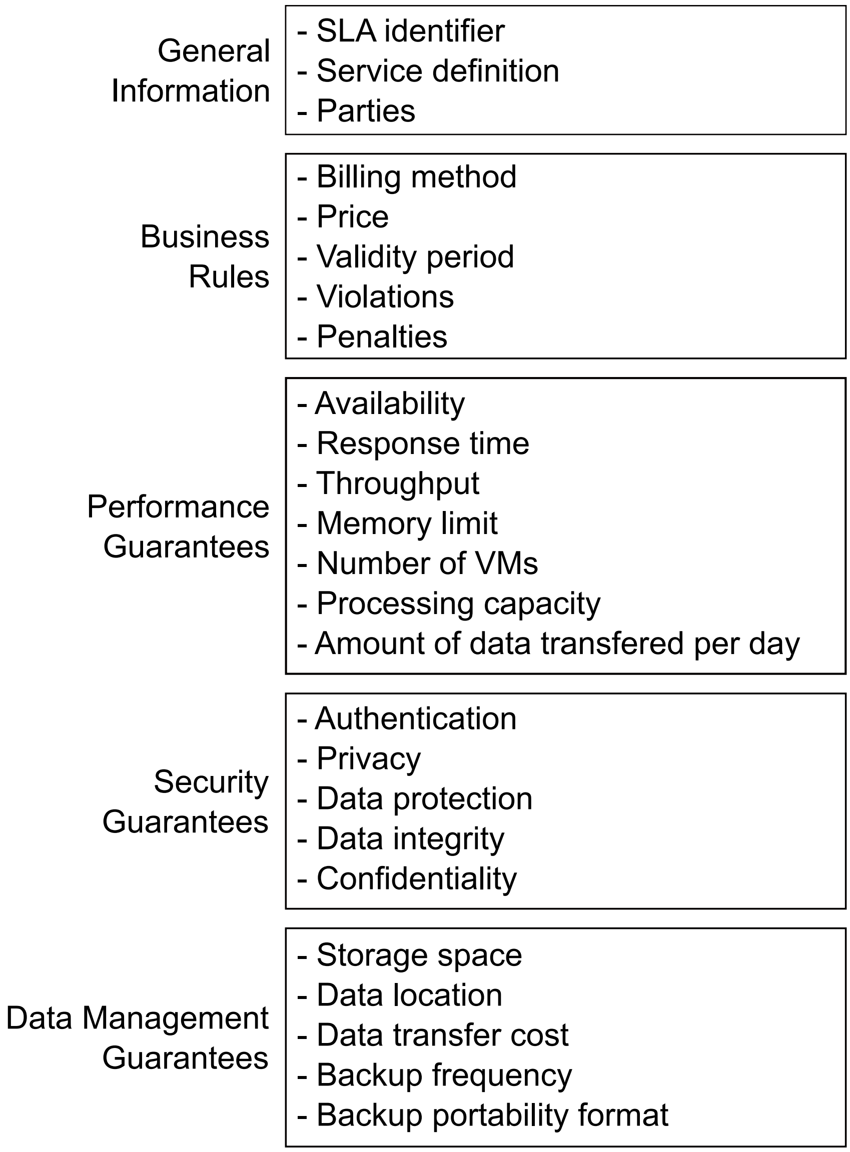

Figure 2: Cloud SLA schema

Service SLA (Figure 3)

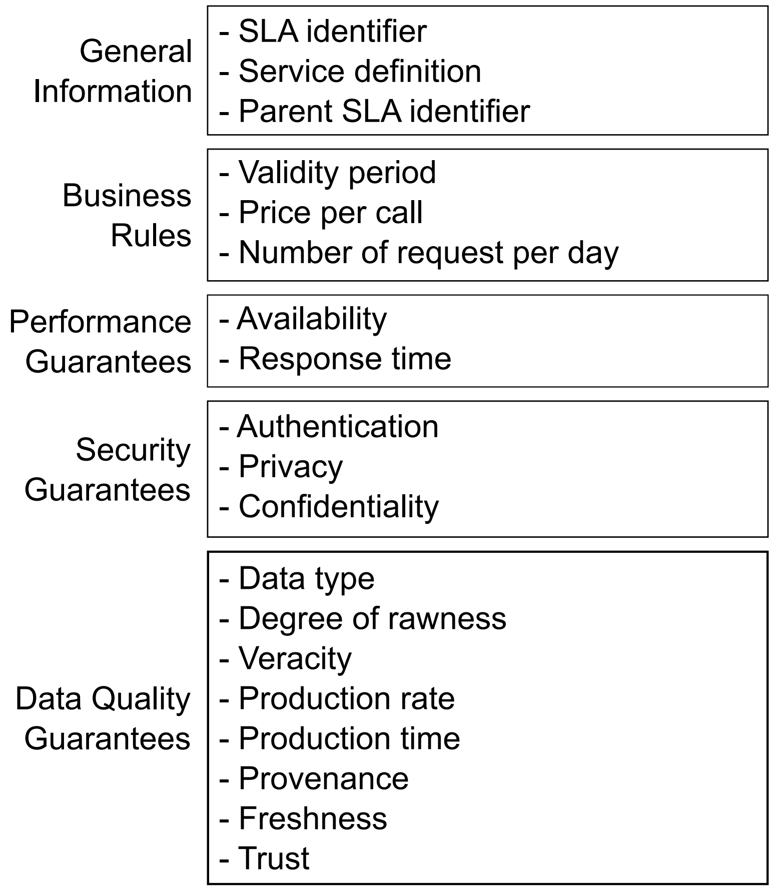

Figure 3: Service SLA schema

Integration SLA

By considering the list of query variations, the following clauses are interesting to be part of the *integration SLA* in order to optimize a further integration request.

1. *The complete query definition.* The query is necessary to identify similarities with a further integration request.

2. *The list of user preferences.* The *user preferences* are necessary to identify similarities with a further integration request.

3. *Integration total cost.* The integration final cost could be a *user preferences* being used a filtering measure.

4. *Integration time.* The necessary time to perform the integration process. This could be a filtering measure considering the *user preferences*.

5. *Used data services.* The *data services* that were used in a previous integration process.

6. *User data consumption environment.* The environment that the user is using to consume the data.

7. *Amount of data transferred.* The amount of data that were transferred and delivered to the user.

8. *Consumption time*. The time necessary to deliver the results to the user considering his consumption environment.

The following information seems to be interest for the integration process. However, it is not an information to be included in the *integration SLA* once it is something general for every integration request. I believe it should be included in a file apart.

***Abstract services* description** (I do not know if it is the best name for it): it is a file that associates an *abstract service* to *data services* that can answer to it including performance information of the *data service*.

This file should include the information regarding the *data services* that can cover an *abstract service*. In the case when no reusable integration exists, this file would avoid the effort of searching for *data services* that can cover an *abstract service* when a new integration request arrives. Even in the case when a reusable integration exists, it could help while identifying and substituting *data services* that are not good in performance or that are not interest considering the current *user preferences*, for instance. Moreover, during the integration process information concerning the processing time of each involved *data service* is collected and included in this file. This information could be interest while choosing a *data services* for a further integration process.