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.

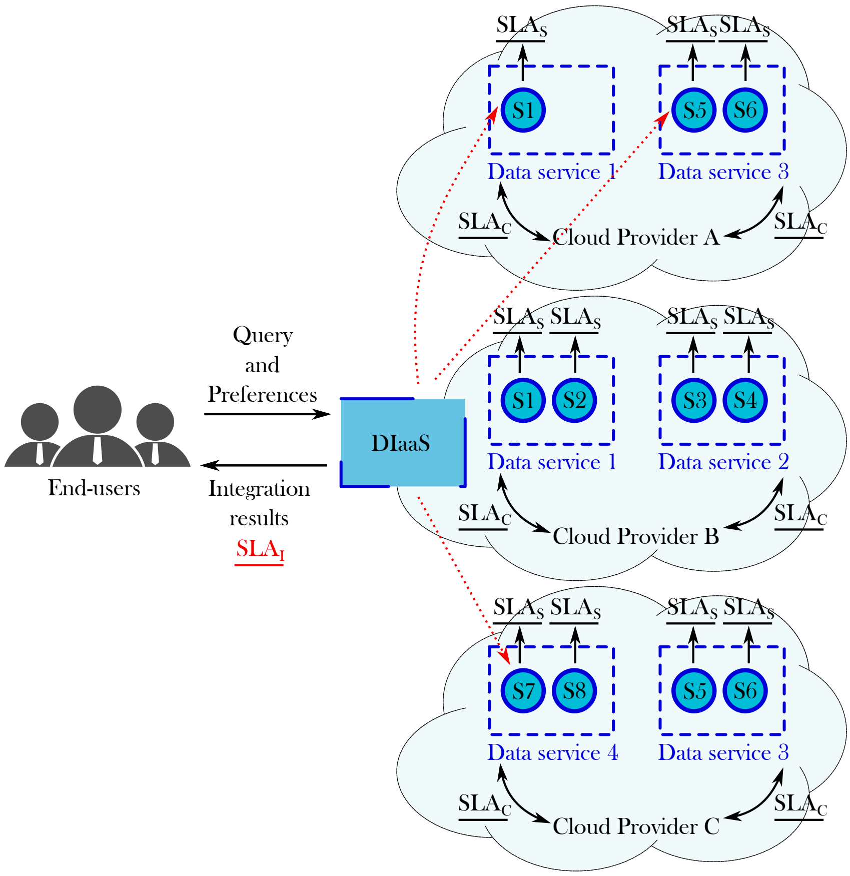

Figure 1: Data integration context overview

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.

1 Basic concepts

A user willing to perform a data integration task defines *(i)* a *query*; and *(ii)* a set of *requirements* over the services or/and over the entire composition (integration). *Queries* are specified in terms of *abstract services*' definitions.

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*; it defines *query* name Q, 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 they are called *head* variables. The right-hand is the *body* definition; it includes a set of *abstract services* followed by a set of *user requirements*. *Abstract services* in the *body* also defines input and output variables. Variables appearing in the *body* are identified by $\overline{I}_l$ and $\overline{O}_l$. *Head* variables can appear in the *body*, and variables appearing only in the *body* are called *local* variables.

In order to be able to compare queries, we assume that there is an ontology (like the one used in [1]) to describe the *abstract services* semantics. The comparison between *queries* is allowed by the query containment concept.

Definition 3 (*Query containment*). A *query* Q_1 is contained in a *query* Q_2, denoted by $Q_1 \subset Q_2$, if and only if the result to Q_1 is a subset of the result to Q_2.

Intuitively, the query equivalence is defined as follows:

Definition 4 (*Query equivalence*). A *query* Q_1 is equivalent to a *query* Q_2, denoted by $Q_1 \equiv Q_2$, if and only if $Q_1 \subset Q_2$ and $Q_2 \subset Q_1$.

Definition 5 (*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 requirement* r could concern *(i)* the entire *query*, in this case noted as r_Q; or *(ii)* a single *service*, noted as r_S. For instance, the *total response time* is obtained by adding the *response time* of each *service* involved in the composition.

Definition 6 (*Requirement domain*). A *requirement domain* is a set of possible values which can be assumed by an *user requirement* r, represented by $Dom(r)$. For instance, a *requirement domain* "*response time*" includes the possible values associated to the *response time user requirement*. Each *user requirement* r_i has its own *requirement domain* D_i.

Definition 7 (*User requirement evaluation*). The evaluation of an *user requirement* r, indicated by $eval(r)$, returns a set of values $\{v_1, .., v_i\}$ that can be assigned to r such that $\{v_1, .., v_i\} \subset Dom(r)$.

Definition 8 (*Comparable requirements*). Given two *user requirements* r_1 and r_2, both can be comparable, denoted by $r_1 \perp r_2$, if and only if: $Dom(r_1) = Dom(r_2)$.

Definition 9 (*User requirements equivalence*). A set of *user requirements* R_1 is equivalent to a set of *user requirements* R_2, represented by $R_1 \equiv R_2$, if and only if: $\forall r_i \in R_1,\ \exists r_j \in R_2 \mid eval(r_i) = eval(r_j)\ and\ |R_1| = |R_2|$.

Definition 10 (*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 $R_1 \ \triangleright\ R_2$, if and only if:

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

Case 2: $\forall r_i \in R_1,\ \exists r_j \in R_1,\ \exists r_k \in R_2,\ \nexists r_l \in R_2 \mid \neg(r_j \perp r_k)\ and\ eval(r_l) \subset eval(r_i)$.

Definition 11 (*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 $R_1 \ \diamond\ R_2$, if and only if:

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

Definition 12 (*User requirements different*). Given a set of *user requirements* R_1 and R_2, R_1 is different of R_2, represented by $R_1 \ \neq\ R_2$, if and only if:

$\forall r_i \in R_1,\ \nexists r_j \in R_2 \mid eval(r_i) \subset eval(r_j)\ or\ eval(r_i) \supset eval(r_j)$.

2 Taxonomy of query variations for promoting the reusability of rewriting results

This section describes the different types of queries which *(i)* can be processed by our integration approach; or *(ii)* can be compared to previous integration requests in order to take advantage from previous integration plans.

Examples illustrate each kind of query and the reusability solution. The table 2 introduces the *abstract services* used in the examples according to our medical scenario. Four *abstract services* are used for defining the queries and *concrete services*.

Example 2.1. To better understand the taxonomy of queries and the reusability solution, let us consider a previous integration request processed by doctor Marcel. *He has searched for patients infected by pneumonia that were treated by doctor Paul using services with availability higher than 97%, response time less than 3 seconds, price per call less than 2 cents, certified data providers, fresh data or not, the overall response time less than 10 seconds and the total cost less than 5 dollars.* This query can be expressed according to the grammar (defined in the section **??**) as follows:

Abstract service	***Description***
$A_1(x?; y!)$	Given a disease x, A_1 returns the list of patients p that were infected by it.
$A_2(z?; w!)$	Given a patient id z, A_2 returns his/her personal information w.
$A_3(d?; y!)$	Given a doctor id d, A_3 returns the list of patients y that were treated by d.
$A_4(h?; y!)$	Given a hospital h, A_4 returns the list of patients y that were treated in it.

$$Q_p\ (disaese?,\ doctor?;\ p\_information!)\ :=\ A_1(disease?;\ p!),\ A_3(doctor?;\ p!),\ A_2(p?;\ p\_information!),\ availability > 97\%,\ response\ time < 3s,\ price\ per\ call < 0.2\$,\ provenance = certified,\ freshness = no,\ total\ response\ time < 10s,\ total\ cost < 5\$$$

The following subsections describe the queries variation and their reusability solution. For all query variation described bellow, we assume a previous integration request defining the *query* Q_p including its *user requirements* R_p as presented in the Example 2.1.

2.1 Case 1: Equivalent *query* and equivalent *user requirements*

Given a user request defining a *query* Q_n and a set of *requirements* R_n, the case in which the *queries* and the *requirements* are equivalent occurs if and only if: $Q_n \equiv Q_p$ and $R_n \equiv R_p$ (Figure 2).

Previous integration request

```
Qp (disease?, doctor?; p_information!) :=
        A1 (disease?; p!),
        A3 (doctor?; p!),
        A2 (p?; p_information!),
        {
          availability > 97%,
          response time < 3s,
          price per call < 0.2$,
          provenance = certified,
          freshness = no,
          total response time < 10s,
          total cost < 5$
        }
```

Incoming integration request

```
Qn (disease?, doctor?; p_information!) :=
        A1 (disease?; p!),
        A3 (doctor?; p!),
        A2 (p?; p_information!),
        {
          availability > 97%,
          response time < 3s,
          price per call < 0.2$,
          provenance = certified,
          freshness = no,
          total response time < 10s,
          total cost < 5$
        }
```

Figure 2: Query case 1

Solution: This is the best case processed by our data integration approach. There are three possible workflows: First, if all involved *concrete services* could be enforced according to their available resources, the previous composition could be re-executed. Then, a new *integration SLA* would be produced and stored to the new request. Second, if part of the involved *concrete services* could not be enforced according to their available resources, these services would be re-allocated by our rewriting algorithm. Once a valid composition is produced, it can be re-executed. Then, a new *integration SLA* would be produced and stored to the new request. Third, if all involved *concrete services* could not be enforced according to their available resources, the new *query* will be dispatched to be rewriting by our rewriting algorithm.

2.2 Equivalent *query* and more restrict *user requirements*

Given a user request defining a *query* Q_n and a set of *requirements* R_n, the case in which the *queries* are equivalent and the *requirements* for the new request are more restrict occurs if and only if: $Q_n \equiv Q_p$ and $R_n \triangleright R_p$ (See figures 3 and 4).

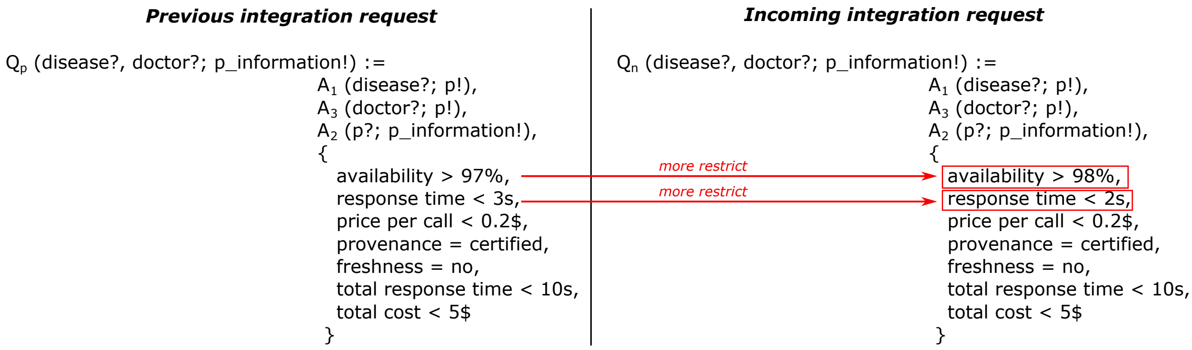

Figure 3: Query case 2a

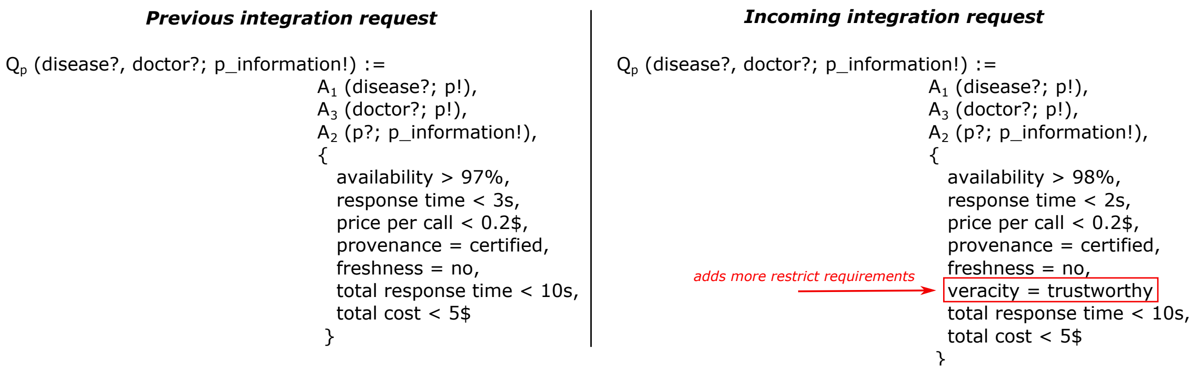

Figure 4: Query case 2b

Solution: There are three possible workflows: First, if all involved *concrete services* could be enforced to the new request according to the new requirements and their available resources, the previous composition could be re-executed. Then, a new *integration SLA* is produced and stored to the new request. Second, if part of the involved *concrete services* could not be enforced to the new request according to the new requirements and their available resources, these services are re-allocated by our rewriting algorithm. Once a new valid composition is produced, it could be executed. Then, a new *integration SLA* is produced and stored to the new request. Third, if all involved *concrete services* could not be enforced to the new request according to the new requirements and their available resources, the new *query* is dispatched to our rewriting algorithm. Once a new valid composition is produced, it could be executed. Then, a new *integration SLA* is produced and stored to the new request.

2.3 Equivalent *query* and less restrict *user requirements*

Given a user request defining a *query* Q_n and a set of *requirements* R_n, the case in which the *queries* are equivalent and the *requirements* for the new request are less restrict occurs if and only if: $Q_n \equiv Q_p$ and $R_p \triangleright R_n$ (See figures 5 and 6).

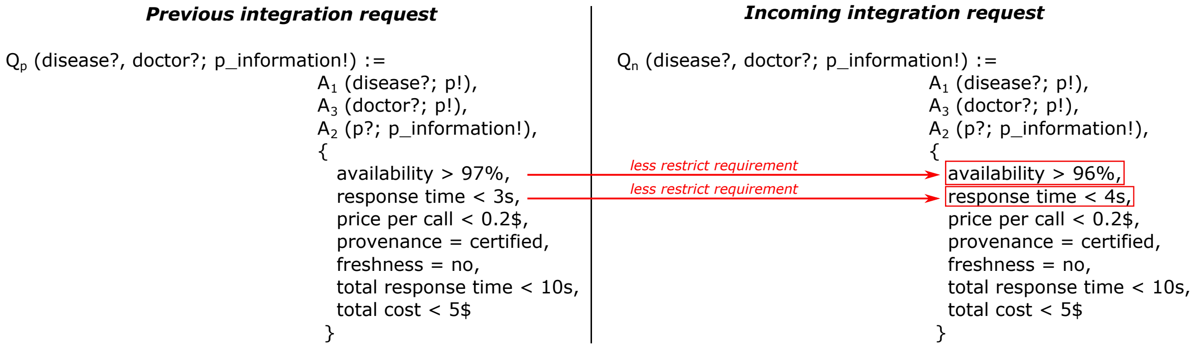

Figure 5: Query case 3a

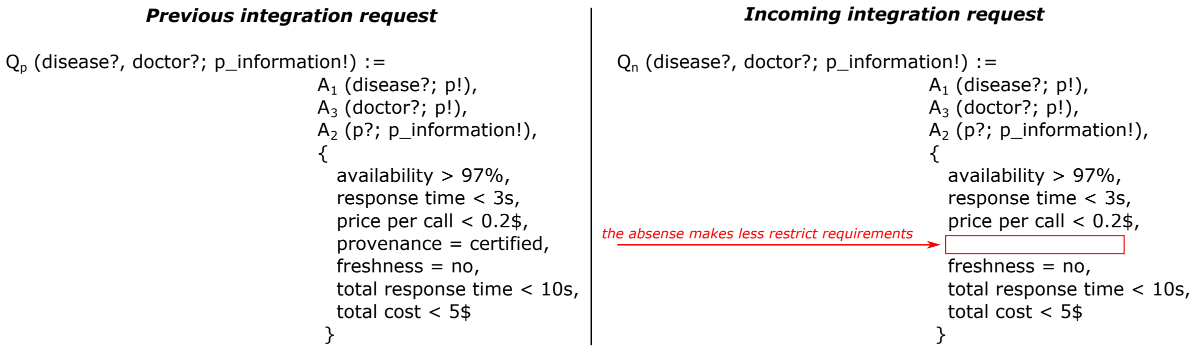

Figure 6: Query case 3b

Solution: There are three possible workflows: First, if all involved *concrete services* could be enforced to the new request according to their available resources, the previous composition could be re-executed. Then, a new *integration SLA* is produced and stored to the new request. Second, if part of the involved *concrete services* could not be enforced to the new request according to their available resources, these services are re-allocated by our rewriting algorithm. Once a new valid composition is produced, it could be executed. Then, a new *integration SLA* is produced and stored to the new request. Third, if all involved *concrete services* could not be enforced to the new request according to their available resources, the new *query* is dispatched to our rewriting algorithm. Once a new valid composition is produced, it could be executed. Then, a new *integration SLA* is produced and stored to the new request.

2.4 Equivalent *queries* and part of the *user requirements* less restrict and part more restrict

Given a user request defining a *query* Q_n and a set of *requirements* R_n, the case in which the the incoming *query* is equivalent to the previous *query* and part of the *requirements* are more restrict and part less restrict occurs if and only if: $Q_n \equiv Q_p$ and $R_n \diamond R_p$ (See figures 7, 8 and 9).

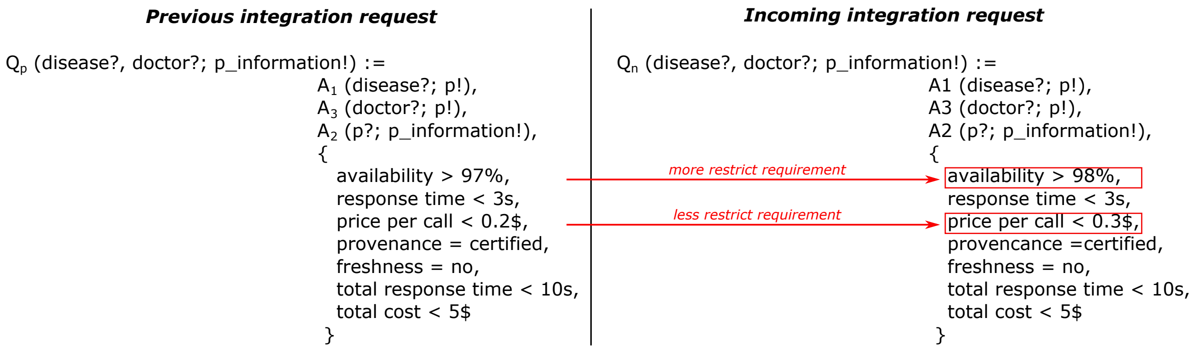

Figure 7: Query case 4a

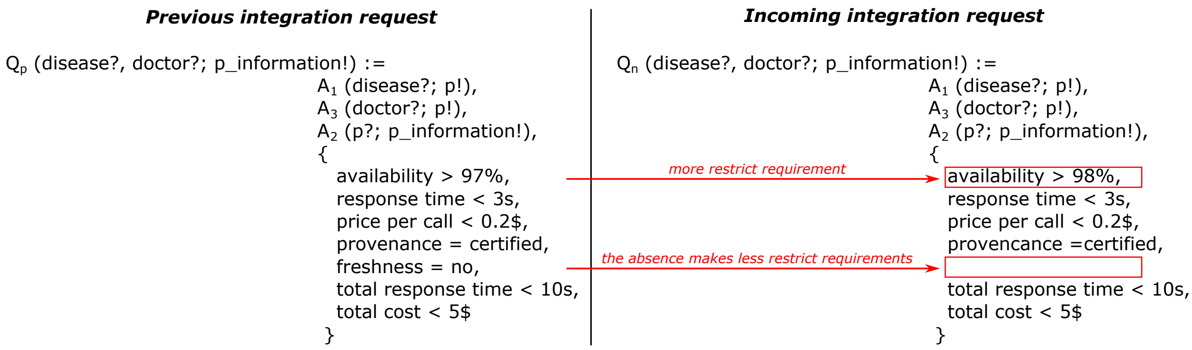

Figure 8: Query case 4b

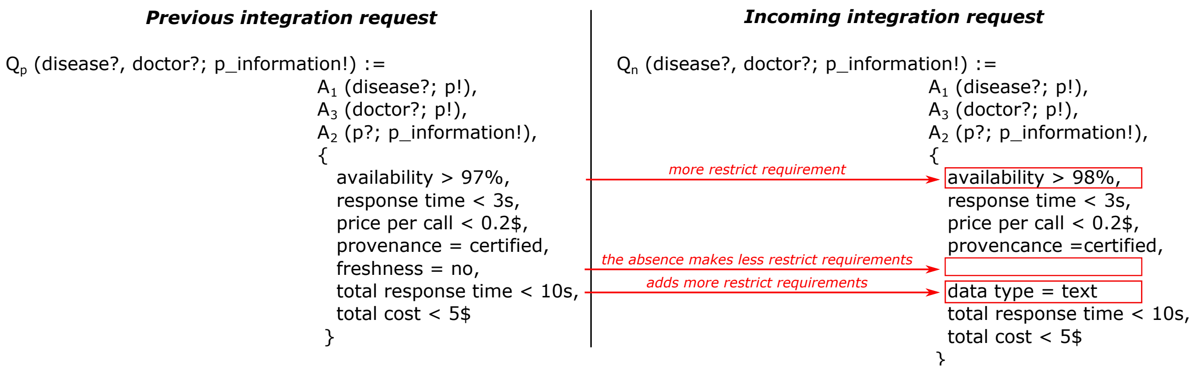

Figure 9: Query case 4c

Solution: There are three possible workflows: First, if all involved *concrete services* could be enforced to the new request according to the new requirements and their available resources, the previous composition could be re-executed. Then, a new *integration SLA* is produced and stored to the new request. Second, if part of the involved *concrete services* could not be enforced to the new request according to the new requirements and their available resources, these services are re-allocated by our rewriting algorithm. Once a new valid composition is produced, it could be executed. Then, a new *integration SLA* is produced and stored to the new request. Third, if all involved *concrete services* could not be enforced to the new request according to the new requirements and their available resources, the new *query* is dispatched to our rewriting algorithm. Once a new valid composition is produced, it could be executed. Then, a new *integration SLA* is produced and stored to the new request.

2.5 Equivalent *query* and different *user requirements*

Given a user request defining a *query* Q_n and a set of *requirements* R_n, the case in which the the incoming *query* is equivalent to the previous *query* and part of the *requirements* are more restrict and part less restrict occurs if and only if: $Q_n \equiv Q_p$ and $R_n \neq R_p$ (See figure 10).

Previous integration request

```
Qp (disease?, doctor?; p_information!) :=
                A1 (disease?; p!),
                A3 (doctor?; p!),
                A2 (p?; p_information!),
                {
                  availability > 97%,
                  response time < 3s,
                  price per call < 0.2$,
                  provenance = certified,
                  freshness = no,
                  total response time < 10s,
                  total cost < 5$
                }
```

Incoming integration request

```
Qn (disease?, doctor?; p_information!) :=
                A1 (disease?; p!),
                A3 (doctor?; p!),
                A2 (p?; p_information!),
                {
adds more restrict requirements --> veracity = trustworthy,
                                    data type = text
                }
```

Figure 10: Query case 5

Solution: There are three possible workflows: First, if all involved *concrete services* could be enforced to the new request according to the new requirements and their available resources, the previous composition could be re-executed. Then, a new *integration SLA* is produced and stored to the new request. Second, if part of the involved *concrete services* could not be enforced to the new request according to the new requirements and their available resources, these services are re-allocated by our rewriting algorithm. Once a new valid composition is produced, it could be executed. Then, a new *integration SLA* is produced and stored to the new request. Third, if all involved *concrete services* could not be enforced to the new request according to the new requirements and their available resources, the new *query* is dispatched to our rewriting algorithm. Once a new valid composition is produced, it could be executed. Then, a new *integration SLA* is produced and stored to the new request.

2.6 *Query* subset and equivalent *user requirements*

Given a user request defining a *query* Q_n and a set of *requirements* R_n, the case in which the the incoming *query* is a subset of the previous *query* and the *requirements* are equivalent occurs if and only if: $Q_n \subset Q_p$ and $R_n \equiv R_p$ (See figure 11).

Solution: There are three possible workflows: First, if all involved *concrete services* could be enforced to the new request according to their available resources, the previous composition could be extended to include the service which are missing to achieve the user results. The rewriting algorithm is responsible to select and include this new service. Once a valid composition is produced, it could be executed. Then, a new *integration*

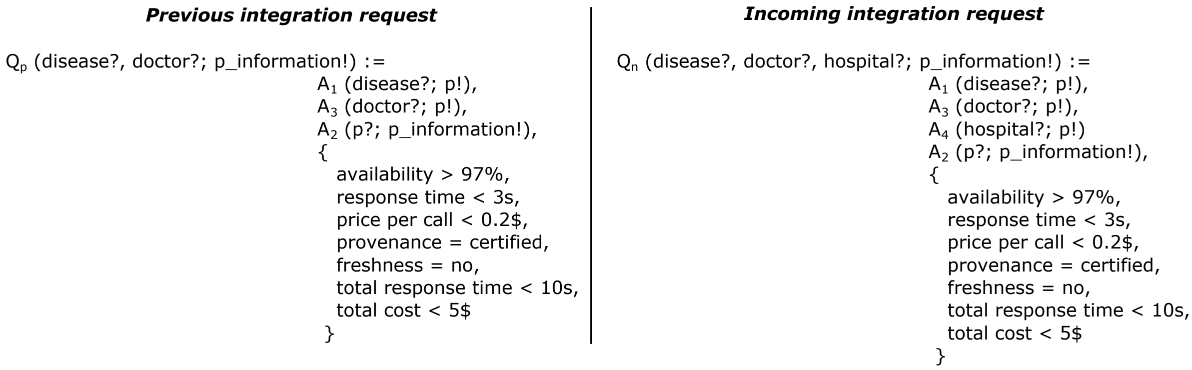

Figure 11: Query case 6

SLA is produced and stored to the new request. Second, if part of the involved *concrete services* could not be enforced to the new request according to their available resources, these services are re-allocated and the services missing to achieve the user needs are included by our rewriting algorithm. Once a new valid composition is produced, it could be executed. Then, a new *integration SLA* is produced and stored to the new request. Third, if all involved *concrete services* could not be enforced to the new request according to their available resources, the new *query* is dispatched to our rewriting algorithm. Once a new valid composition is produced, it could be executed. Then, a new *integration SLA* is produced and stored to the new request.

2.7 *Query* subset and more restrict *user requirements*

Given a user request defining a *query* Q_n and a set of *requirements* R_n, the case in which the the incoming *query* is a subset of the previous *query* and the *requirements* for the new request are more restrict occurs if and only if: $Q_n \subset Q_p$ and $R_n \triangleright R_p$ (See figures 12 and 13).

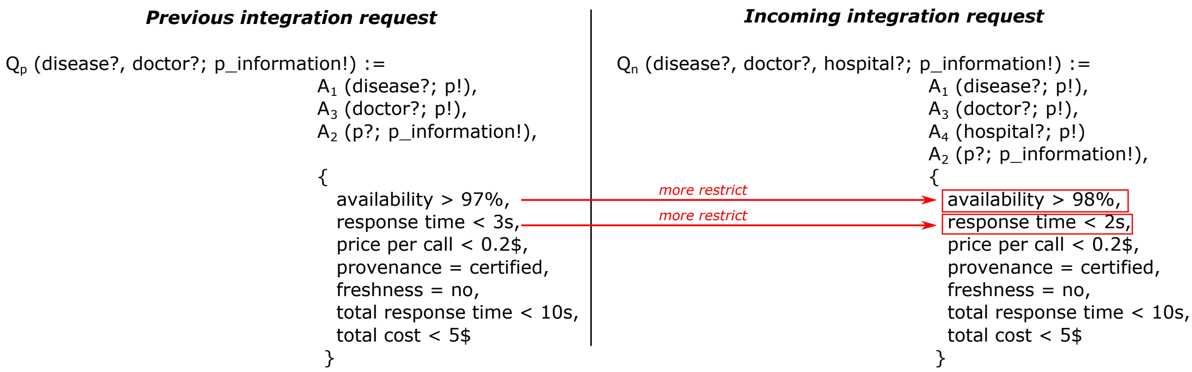

Figure 12: Query case 7a

Solution: There are three possible workflows: First, if all involved *concrete services* could be enforced to the new request according to the new *requirements* and their available resources, the previous composition could be extended to include the services which are missing to achieve the user results. The rewriting algorithm is responsible to select and include new services. Once a valid composition is produced, it could be executed. Then,

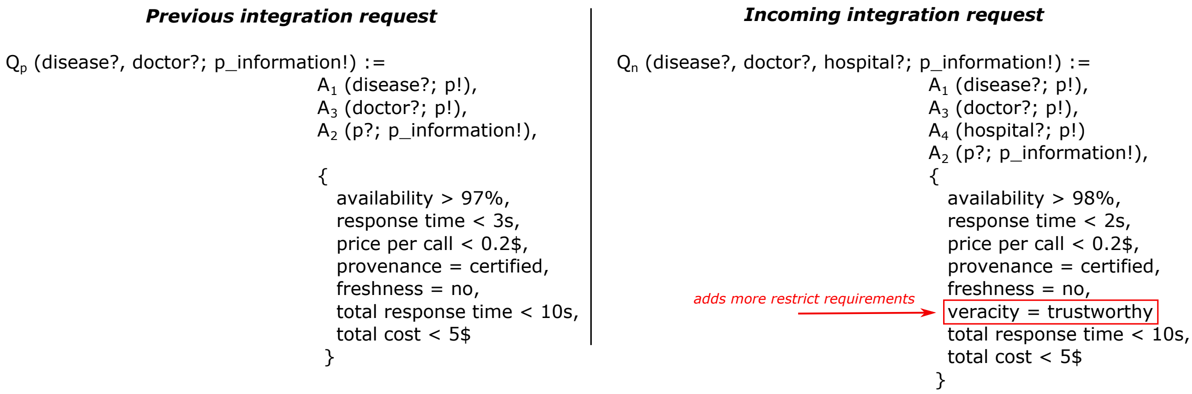

Figure 13: Query case 7b

a new *integration SLA* is produced and stored to the new request. Second, if part of the involved *concrete services* could not be enforced to the new request according to the new *requirements* and their available resources, these services are re-allocated and the services missing to achieve the user needs are included by our rewriting algorithm. Once a new valid composition is produced, it could be executed. Then, a new *integration SLA* is produced and stored to the new request. Third, if all involved *concrete services* could not be enforced to the new request according to the new *requirements* and their available resources, the new *query* is dispatched to our rewriting algorithm. Once a new valid composition is produced, it could be executed. Then, a new *integration SLA* is produced and stored to the new request.

2.8 *Query* subset and less restrict *user requirements*

Given a user request defining a *query* Q_n and a set of *requirements* R_n, the case in which the the incoming *query* is a subset of the previous *query* and the *requirements* for the new request are less restrict occurs if and only if: $Q_n \subset Q_p$ and $R_p \triangleright R_n$ (See figures 14 and 15).

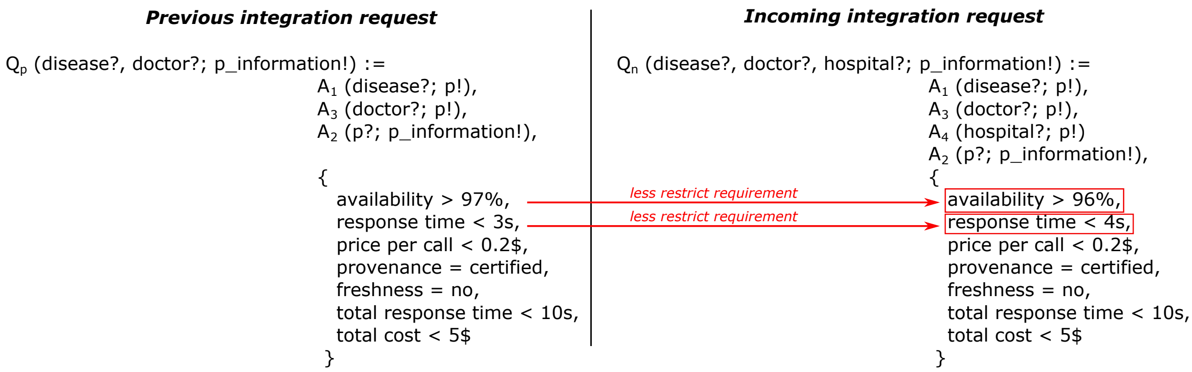

Figure 14: Query case 8a

Solution: There are three possible workflows: First, if all involved *concrete services* could be enforced to the new request according to their available resources, the previous composition could be extended to include the services which are missing to achieve the

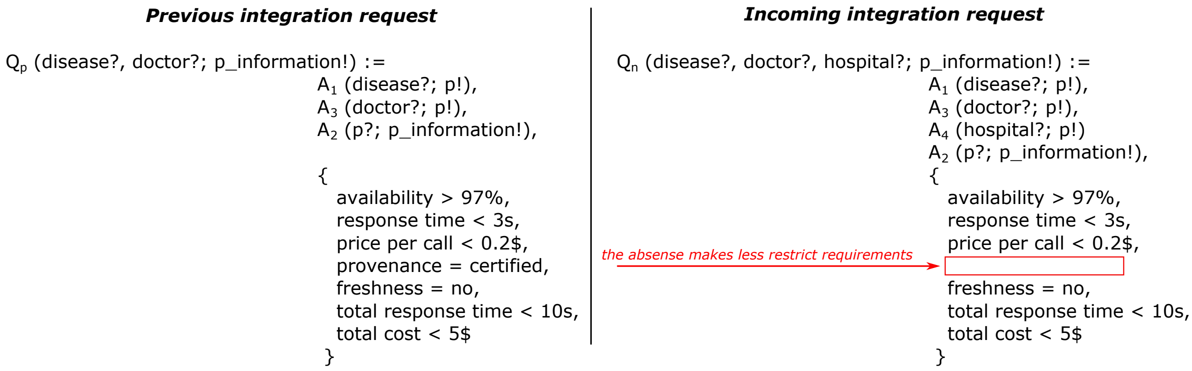

Figure 15: Query case 8b

user results. The rewriting algorithm is responsible to select and include new services. Once a valid composition is produced, it could be executed. Then, a new *integration SLA* is produced and stored to the new request. Second, if part of the involved *concrete services* could not be enforced to the new request according to their available resources, these services are re-allocated and the services missing to achieve the user needs are included by our rewriting algorithm. Once a new valid composition is produced, it could be executed. Then, a new *integration SLA* is produced and stored to the new request. Third, if all involved *concrete services* could not be enforced to the new request according to their available resources, the new *query* is dispatched to our rewriting algorithm. Once a new valid composition is produced, it could be executed. Then, a new *integration SLA* is produced and stored to the new request.

2.9 *Query* subset and part of the *user requirements* less restrict and part more restrict

Given a user request defining a *query* Q_n and a set of *requirements* R_n, the case in which the the incoming *query* is a subset of the previous *query* and the *requirements* for the new request are less restrict occurs if and only if: $Q_n \subset Q_p$ and $R_p \diamond R_n$ (See figures 16 and 17, 18).

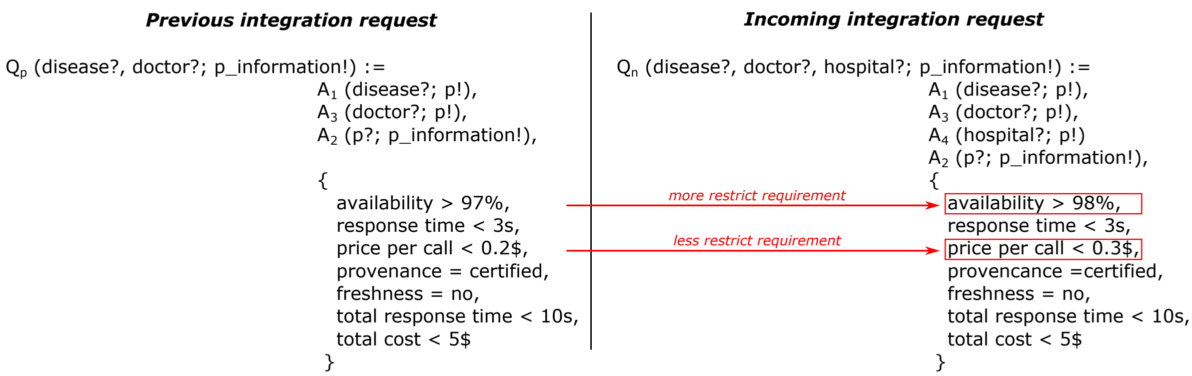

Figure 16: Query case 9a

Solution: There are three possible workflows: First, if all involved *concrete services* could be enforced to the new request according to the new *requirements* and their available

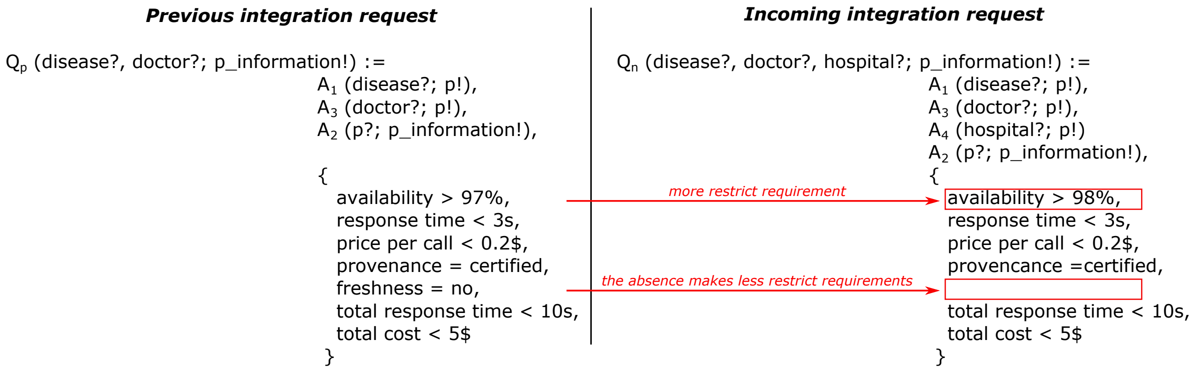

Figure 17: Query case 9b

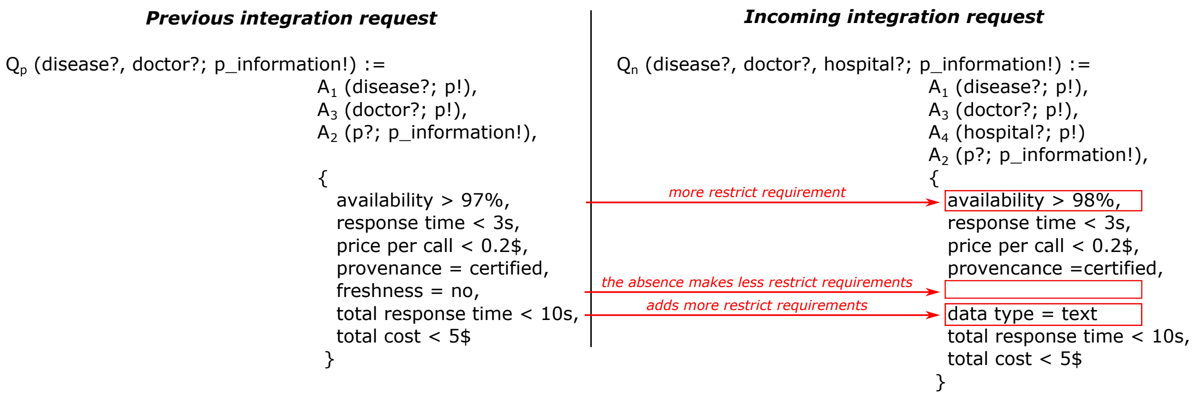

Figure 18: Query case 9c

resources, the previous composition could be extended to include the services which are missing to achieve the user results. The rewriting algorithm is responsible to select and include new services. Once a valid composition is produced, it could be executed. Then, a new *integration SLA* is produced and stored to the new request. Second, if part of the involved *concrete services* could not be enforced to the new request according to the new *requirements* and their available resources, these services are re-allocated and the services missing to achieve the user needs are included by our rewriting algorithm. Once a new valid composition is produced, it could be executed. Then, a new *integration SLA* is produced and stored to the new request. Third, if all involved *concrete services* could not be enforced to the new request according to the new *requirements* and their available resources, the new *query* is dispatched to our rewriting algorithm. Once a new valid composition is produced, it could be executed. Then, a new *integration SLA* is produced and stored to the new request.

2.10 *Query* subset and different *user requirements*

Given a user request defining a *query* Q_n and a set of *requirements* R_n, the case in which the the incoming *query* is a subset of the previous *query* and the *requirements* for the new request are less restrict occurs if and only if: $Q_n \subset Q_p$ and $R_p \neq R_n$ (See figure 19).

Solution: There are three possible workflows: First, if all involved *concrete services* could be enforced to the new request according to the new *requirements* and their available

Previous integration request

```
Qp (disease?, doctor?; p_information!) :=
        A1 (disease?; p!),
        A3 (doctor?; p!),
        A2 (p?; p_information!),

        {
          availability > 97%,
          response time < 3s,
          price per call < 0.2$,
          provenance = certified,
          freshness = no,
          total response time < 10s,
          total cost < 5$
        }
```

Incoming integration request

```
Qn (disease?, doctor?, hospital?; p_information!) :=
        A1 (disease?; p!),
        A3 (doctor?; p!),
        A4 (hospital?; p!)
        A2 (p?; p_information!),
        {
adds more restrict requirements
          veracity = trustworthy,
          data type = text
        }
```

Figure 19: Query case 10

resources, the previous composition could be extended to include the services which are missing to achieve the user results. The rewriting algorithm is responsible to select and include new services. Once a valid composition is produced, it could be executed. Then, a new *integration SLA* is produced and stored to the new request. Second, if part of the involved *concrete services* could not be enforced to the new request according to the new *requirements* and their available resources, these services are re-allocated and the services missing to achieve the user needs are included by our rewriting algorithm. Once a new valid composition is produced, it could be executed. Then, a new *integration SLA* is produced and stored to the new request. Third, if all involved *concrete services* could not be enforced to the new request according to the new *requirements* and their available resources, the new *query* is dispatched to our rewriting algorithm. Once a new valid composition is produced, it could be executed. Then, a new *integration SLA* is produced and stored to the new request.

2.11 *Query* superset and equivalent *user requirements*

Given a user request defining a *query* Q_n and a set of *requirements* R_n, the case in which the the incoming *query* is a superset of the previous *query* and the *requirements* are equivalent occurs if and only if: $Q_p \subset Q_n$ and $R_n \equiv R_p$ (See figure 20).

Previous integration request

```
Qp (disease?, doctor?; p_information!) :=
        A1 (disease?; p!),
        A3 (doctor?; p!)
        A2 (p?; p_information!),
        {
          availability > 97%,
          response time < 3s,
          price per call < 0.2$,
          provenance = certified,
          freshness = no,
          total response time < 10s,
          total cost < 5$
        }
```

Incoming integration request

```
Qn (disease?; p_information!) :=
        A1 (disease?; p!),
        A2 (p?; p_information!),

        {
          availability > 97%,
          response time < 3s,
          price per call < 0.2$,
          provenance = certified,
          freshness = no,
          total response time < 10s,
          total cost < 5$
        }
```

Figure 20: Query case 11

Solution: There are three possible solutions for this case. First, if the services used in Q_p which are interesting for the *query* Q_n could be correctly mapped and enforced according to the resources available, they could be composed and executed. Second, if part of the services used in Q_p which are interesting for the *query* Q_n could be correctly mapped and enforced according to the resources available, it will be reused. However, the other services necessary to achieve the results will be selected and composed using our rewriting algorithm. Then, when the final composition is produced in accordance with the new *requirements*, it can be executed. And, third, if none of the services used in Q_p which are interesting for the *query* Q_n could be correctly mapped and enforced according to the resources available, the new *query* will be dispatched to our rewriting algorithm to produce a valid rewriting for this request.

2.12 *Query* superset and more restrict *user requirements*

Given a user request defining a *query* Q_n and a set of *requirements* R_n, the case in which the the incoming *query* is a superset of the previous *query* and the *requirements* for the new request are more restrict occurs if and only if: $Q_p \subset Q_n$ and $R_n \triangleright R_p$ (See figures 21 and 22.

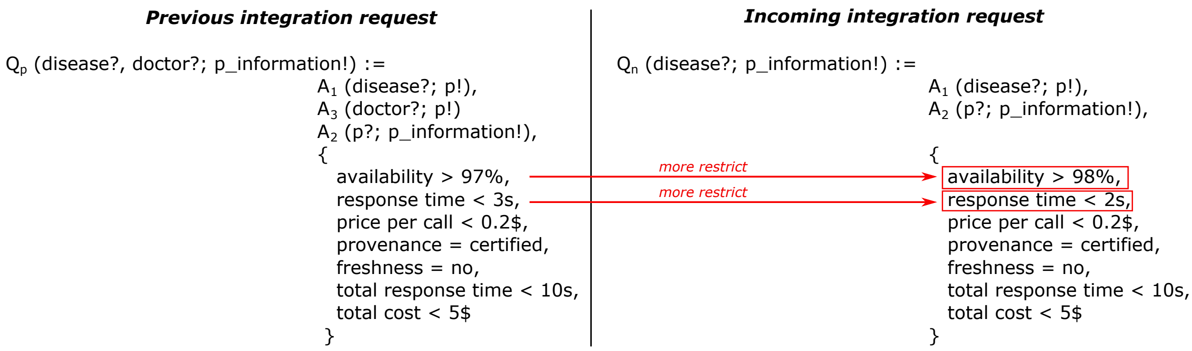

Figure 21: Query case 12a

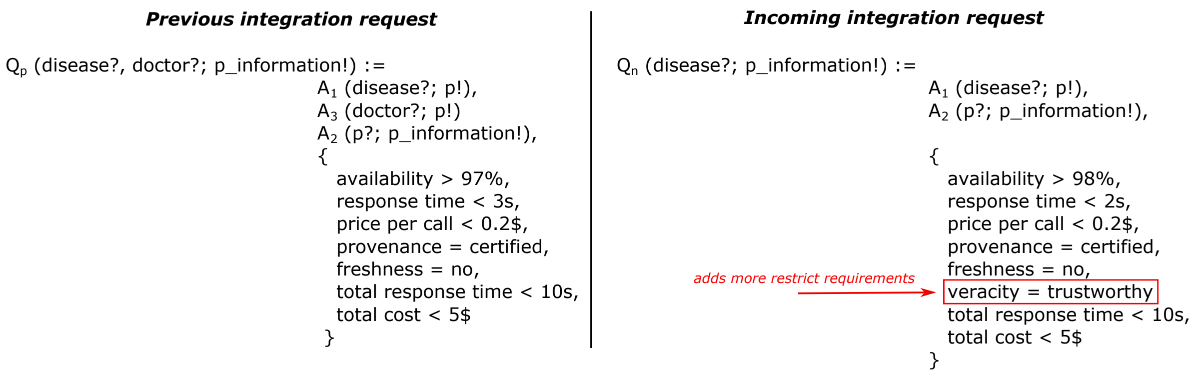

Figure 22: Query case 12b

Solution: There are three possible solutions for this case. First, if the services used in Q_p which are interesting for the *query* Q_n could be correctly mapped and enforced according to the new *requirements* and the resources available, they could be composed and executed. Second, if part of the services used in Q_p which are interesting for the *query*

Q_n could be correctly mapped and enforced according to the new *requirements* and the resources available, it will be reused. However, the other services necessary to achieve the results will be selected and composed using our rewriting algorithm. Then, when the final composition is produced in accordance with the new *requirements*, it can be executed. And, third, if none of the services used in Q_p which are interesting for the *query* Q_n could be correctly mapped and enforced according to the new *requirements* and the resources available, the new *query* will be dispatched to our rewriting algorithm to produce a valid rewriting for this request.

2.13 *Query* superset and less restrict *user requirements*

Given a user request defining a *query* Q_n and a set of *requirements* R_n, the case in which the the incoming *query* is a superset of the previous *query* and the *requirements* for the new request are more restrict occurs if and only if: $Q_p \subset Q_n$ and $R_p \triangleright R_n$ (See figures 23 and 24).

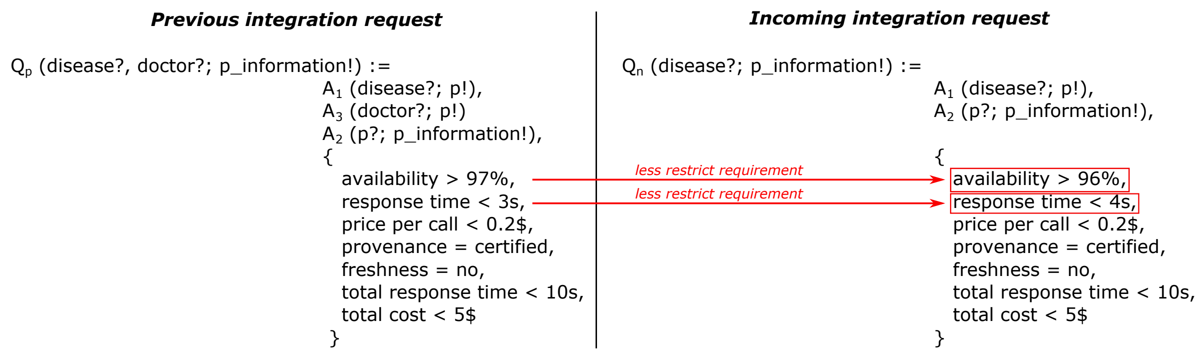

Figure 23: Query case 13a

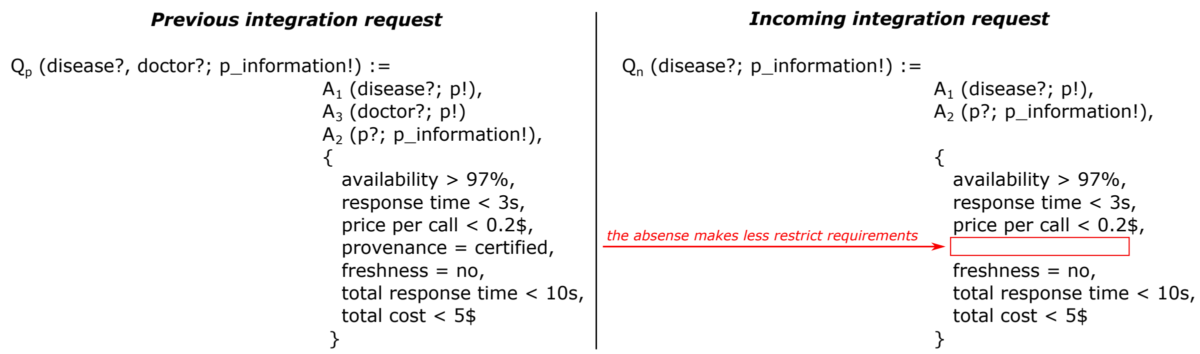

Figure 24: Query case 13b

Solution: There are three possible solutions for this case. First, if the services used in Q_p which are interesting for the *query* Q_n could be correctly mapped and enforced according to the resources available, they could be composed and executed. Second, if part of the services used in Q_p which are interesting for the *query* Q_n could be correctly mapped and enforced according to the resources available, it will be reused. However, the other services necessary to achieve the results will be selected and composed using our rewriting algorithm. Then, when the final composition is produced in accordance with the new *requirements*, it can be executed. And, third, if none of the services used in Q_p which are interesting for the *query* Q_n could be correctly mapped and enforced according to the resources available, the new *query* will be dispatched to our rewriting algorithm to produce a valid rewriting for this request.

2.14 *Query* superset and part of the *user requirements* less restrict and part more restrict

Given a user request defining a *query* Q_n and a set of *requirements* R_n, the case in which the the incoming *query* is a superset of the previous *query* and the *requirements* for the new request are more restrict occurs if and only if: $Q_p \subset Q_n$ and $R_p \diamond R_n$. The figures 25, 26 and 27 illustrate the different variation of *requirements*.

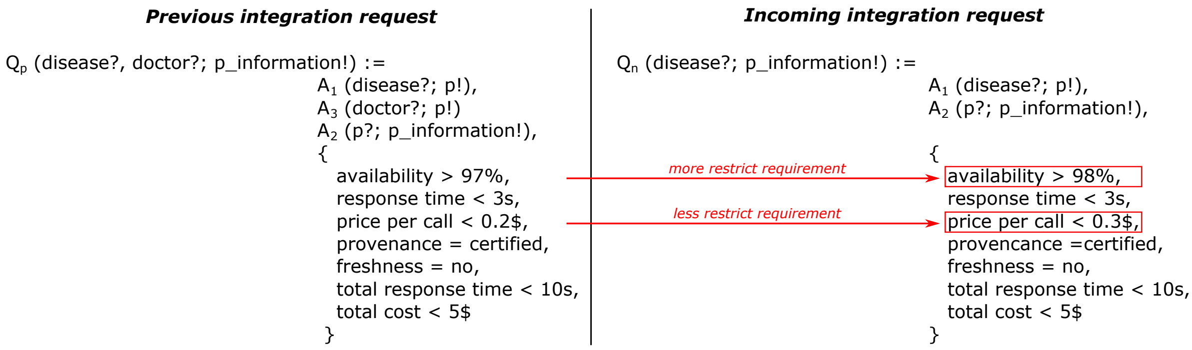

Figure 25: Query case 14a

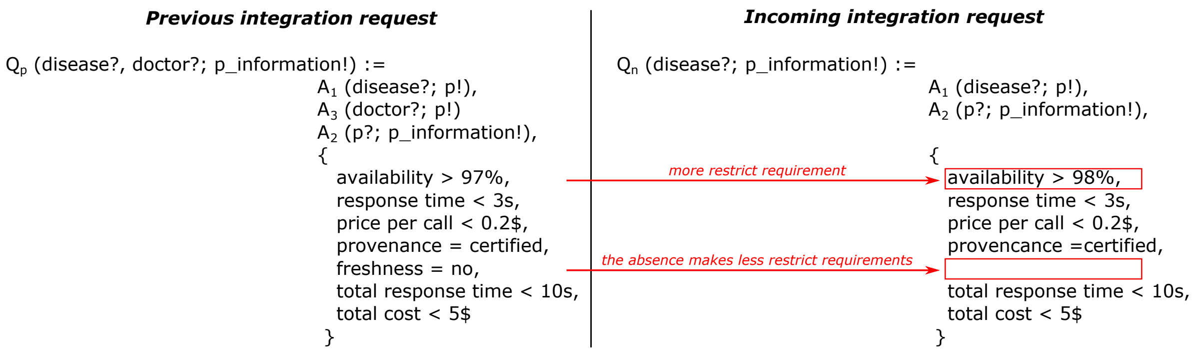

Figure 26: Query case 14b

Solution: There are three possible solutions for this case. First, if the services used in Q_p which are interesting for the *query* Q_n could be correctly mapped and enforced according to the new *requirements* and the resources available, they could be composed and executed. Second, if part of the services used in Q_p which are interesting for the *query* Q_n could be correctly mapped and enforced according to the new *requirements* and the resources available, it will be reused. However, the other services necessary to achieve the

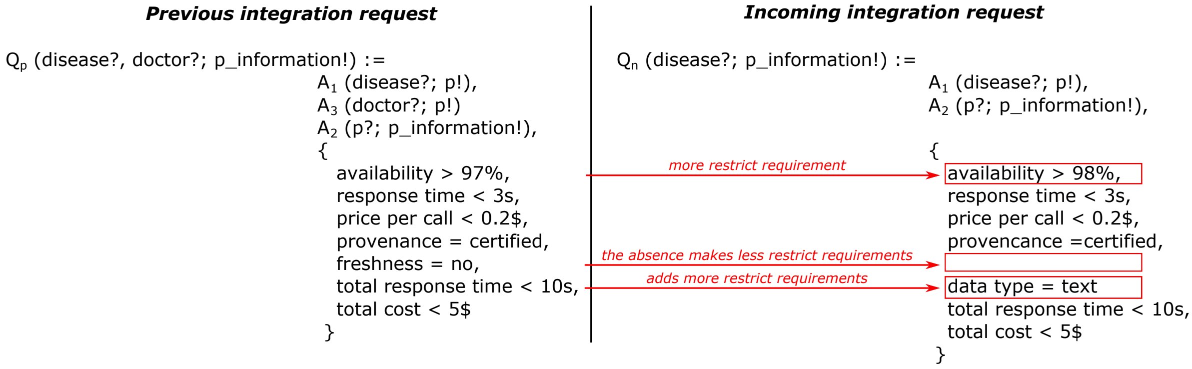

Figure 27: Query case 14c

results will be selected and composed using our rewriting algorithm. Then, when the final composition is produced in accordance with the new *requirements*, it can be executed. And, third, if none of the services used in Q_p which are interesting for the *query* Q_n could be correctly mapped and enforced according to the new *requirements* and the resources available, the new *query* will be dispatched to our rewriting algorithm to produce a valid rewriting for this request.

2.15 *Query* superset and different *user requirements*

I think that this case can be included as one more sub-case of queries with more restrict requirements. Given a user request defining a *query* Q_n and a set of *requirements* R_n, the case in which the the incoming *query* is a superset of the previous *query* and the *requirements* for the new request are more restrict occurs if and only if: the result of the previous *query* is contained in the result of the incoming *query* ($Q_p \subset Q_n$) and the set of requirements R_n defines only requirements that are not present in the set of requirement R_p (see figure 28).

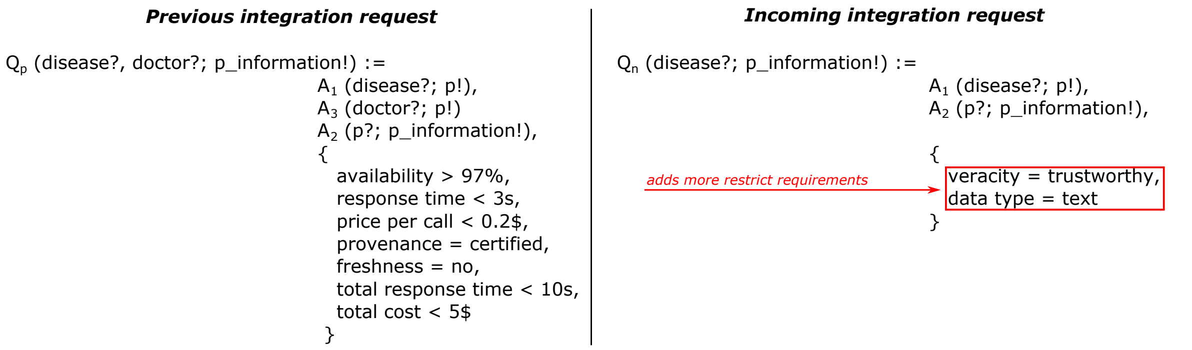

Figure 28: Query case 15

Solution: There are three possible solutions for this case. First, if the services used in Q_p which are interesting for the *query* Q_n could be correctly mapped and enforced according to the new *requirements* and the resources available, they could be composed and executed. Second, if part of the services used in Q_p which are interesting for the *query* Q_n could be correctly mapped and enforced according to the new *requirements* and the resources available, it will be reused. However, the other services necessary to achieve the results will be selected and composed using our rewriting algorithm. Then, when the final composition is produced in accordance with the new *requirements*, it can be executed. And, third, if none of the services used in Q_p which are interesting for the *query* Q_n could be correctly mapped and enforced according to the new *requirements* and the resources available, the new *query* will be dispatched to our rewriting algorithm to produce a valid rewriting for this request.

2.16 Different *queries*

The last query variation is the worst case processed by our data integration approach. It occurs when the queries are completely different. This means that the expected result of the new *query* Q_n can not be compared with the result of any other previous integration request Q_p. In such situation, there is no reusability solution. The new *query* should be dispatched to our rewriting algorithm (presented in the section **??**) to produce a service composition according to the user needs.

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 29)

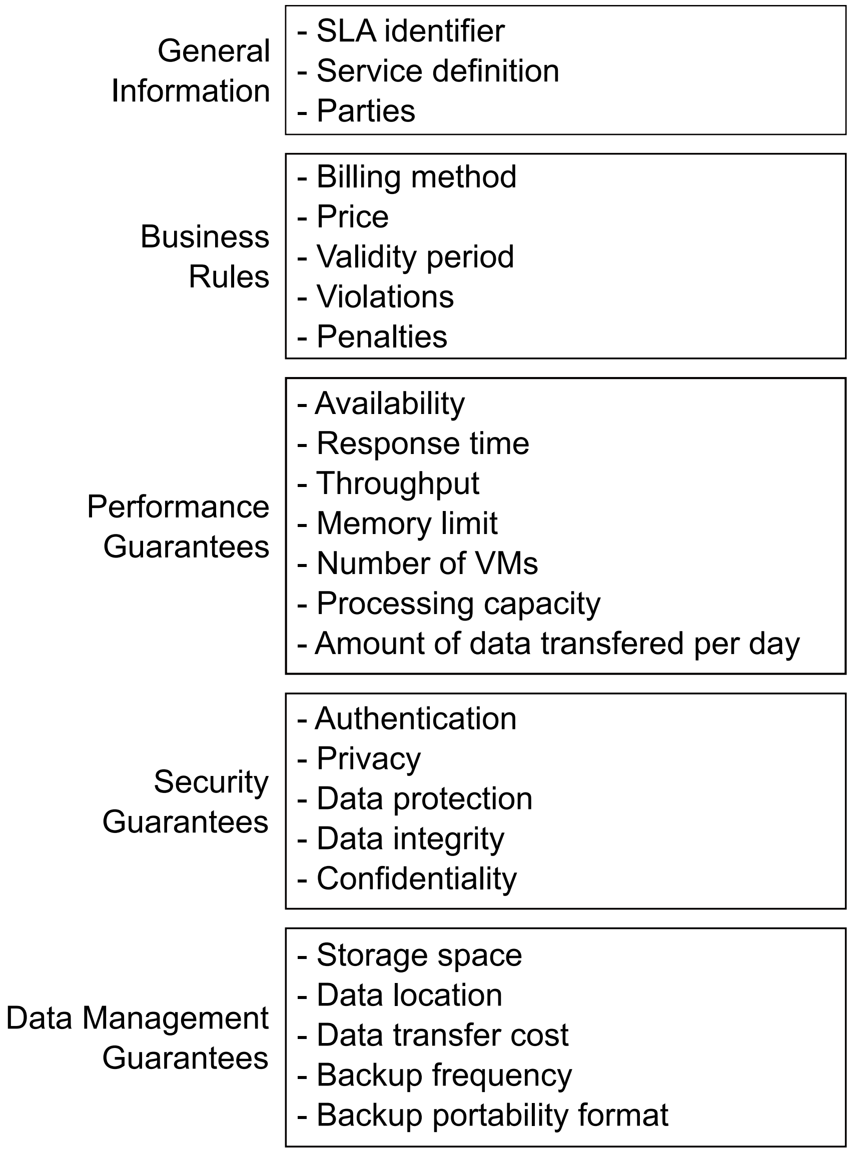

Figure 29: Cloud SLA schema

Service SLA (Figure 30)

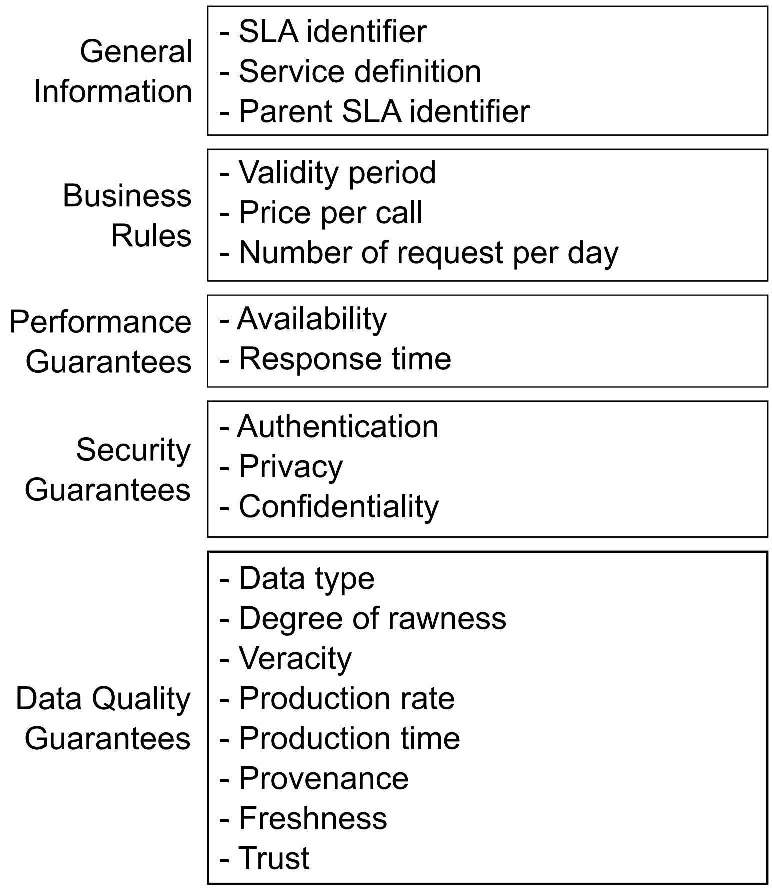

Figure 30: Service SLA schema

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.

References

[1] M. Barhamgi, D. Benslimane, and B. Medjahed. A query rewriting approach for web service composition. *Services Computing, IEEE Transactions on*, 3(3):206–222, July 2010.