Intégration de données basée sur la qualité pour l'enrichissement des sources de données locales dans le Service Lake

Hiba Alili

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à l'université Paris-Dauphine et l'Université de la Manouba

Quality-based Data Integration for Enriching User Data Sources in Service Lakes

Soutenue par
Hiba ALILI
Le **27/11/2019**

École doctorale n°543
ED de Dauphine

Spécialité
Informatique

Composition du jury :

Djamel BENSLIMANE Professeur des Universités, Université Claude Bernard Lyon 1	*Rapporteur*
Genoveva VARGAS-SOLAR Professeur, Université Grenoble-Alpes	*Rapporteur*
Zoubida KEDAD Professeur, Université de Versailles	*Examinateur*
Dimitris KOTZINOS Professeur, Université Cergy-Pontoise	*Président du Jury*
Daniela GRIGORI Professeur, Université Paris Dauphine	*Directeur*
Henda HAJJAMI BEN GHEZALA Professeur, Université de la Manouba	*Directeur*
Khalid BELHAJJAME Maître de Conférences, Université Paris Dauphine	*Co-encadrant*
Rim DRIRA Maître Assistant, Université de la Manouba	*Co-encadrant*

DAUPHINE UNIVERSITÉ PARIS | PSL | ENSI

Acknowledgements

This dissertation would not exist without the support of many people over four years.

Foremost, I would like to express my sincere gratitude to my supervisors Khalid Belhajjame, Rim Drira, Daniela Grigori and Henda Hajjami Ben Ghezala for allowing me to carry out this thesis. Thank you for your patience in supervising this work, your advises, your insightful comments and for the many hours spent helping this work. Thank you for your endless patience in improving my writing, for your comments on chapter drafts and for your great efforts to explain things clearly and simply.

My humble gratitude goes to the faculty members of the LAMSADE lab for all of their scientific support, friendship and encouragement. Particularly, I would like to thank Juliette Rouchier, Joyce El Haddad and Furini Fabio for following my advancement during each year of my thesis.

I thank my fellow labmates in LAMSADE and RIADI for the stimulating discussions, for the sleepless nights we were working together before deadlines, for their encouragement and moral support, and for all good and bad times we get together. A particular acknowledgment goes to to all my outside-the-university friends for their friendship, their support and the important role they play in my life.

Lastly, and most importantly, I would like to thank my parents, my brothers for their steadfast support, their love and patience. I dedicate this work to them.

Abstract

In the Big Data era, companies are moving away from traditional data-warehouse solutions whereby expensive and time-consuming ETL (Extract, Transform, Load) processes are used, towards data lakes in order to manage their increasingly growing data. Yet the stored knowledge in companies' databases, even though in the constructed data lakes, can never be complete and up-to-date, because of the continuous production of data. Local data sources often need to be augmented and enriched with information coming from external data sources. Unfortunately, the data enrichment process is one of the manual labors undertaken by experts who enrich data by adding information based on their expertise or select relevant data sources to complete missing information. Such work can be tedious, expensive and time-consuming, making it very promising for automation.

This thesis presents an active user-centric data integration approach to automatically enrich local data sources, in which the missing information is leveraged on the fly from web sources using data services. Accordingly, our approach enables users to query for information about concepts that are not defined in the data source schema. In doing so, we take into consideration a set of user preferences such as the cost threshold and the response time necessary to compute the desired answers, while ensuring a good quality of the obtained results.

The first part of this thesis describes the enrichment of data sources' schemas with the concepts required for the processing of users' data queries. To do so, we have developed two algorithms. The first allows to identify the missing data, more specifically the missing concepts and associated attributes that are required by the user queries, but that are not provided by his data source. The second algorithm enriches the schema of the user data source by adding the missing elements determined by the first algorithm. In a second part of the thesis, we show how the Service Lake can be leveraged to enrich local datasets. We propose a new quality-based service composition approach to identify the relevant data services that can be used to populate the miss-

ing information. In doing so, we adapted local-as-view data integration techniques. Moreover, we elaborated a knapsack-based algorithm to select the services that yield good quality results without exceeding a given budget (time and monetary cost). The retrieved information is seamlessly and transparently integrated into the local dataset.

The following part addresses the structural and semantic heterogeneities that may exist between the data types provided by different data services and the data types stored in local data sources. This problem presents one of the major issues faced while selecting and composing the relevant data services for answering users' queries. We propose to define views for all the data services available in the Service Lake over the relations in the local data source schema. The first step relies on the matching results computed by COMA++ between the schema of the local data source and the input/output parameters of the data service. Using the matching obtained in the first step, the second step automatically creates a node/edge-weighted graph, depicting the schema of the local data source such that the weights of the nodes represent an aggregated matching score on the node attributes. Then, we study finding the top-k minimum cost connected trees that contain all service parameters at least once in the graph. We do so by exploring Steiner trees.

Last but not least, our contribution, MoDaaS, presents a model-driven framework for the modeling and the description of data services and DaaS services in particular. We developed MoDaaS to encourage providers to adopt a standard model for the modeling of their services' capabilities and concerns according to a shared ontology, thus enabling them to automatically generate service views in order to assist the integration and data exchange between heterogeneous services.

Key words : User-Centric Data Integration, Data Provisioning Service Lakes, Schema Enriching, Data Services, Data as a Service(DaaS), Service Views, Composition, Orchestration, Data Quality, Quality of Service(QoS), Query Processing, User preferences, Steiner Trees, Cloud Computing, Semantic Annotation, Domain Ontologies, Model Driven Engineering(MDE), Reuse and Specialization

Résumé

De nos jours, d'énormes volumes de données sont créés en continu et les utilisateurs s'attendent à ce que ceux-ci soient collectés, stockés et traités quasiment en temps réel. Ainsi, les lacs de données sont devenus une solution attractive par rapport aux entrepôts de données classiques coûteux et fastidieux (nécessitant une démarche ETL), pour les entreprises qui souhaitent stocker leurs données. Malgré leurs volumes, les données stockées dans les lacs de données des entreprises sont souvent incomplètes voire non mises à jour vis-à-vis des besoins (requêtes) des utilisateurs. Les sources de données locales ont donc besoin d'être enrichies. Par ailleurs, la diversité et l'expansion du nombre de sources d'information disponibles sur le web a rendu possible l'extraction des données en temps réel. Ainsi, afin de permettre d'accéder et de récupérer l'information de manière simple et interopérable, les sources de données sont de plus en plus intégrées dans les services Web. Il s'agit plus précisément des services de données, y compris les services DaaS (Data as a Service) du Cloud Computing. L'enrichissement manuel des sources locales implique plusieurs tâches fastidieuses telles que l'identification des services pertinents, l'extraction et l'intégration de données hétérogènes, la définition des mappings service-source, etc.

Dans un tel contexte, nous proposons une nouvelle approche d'intégration de données centrée utilisateur. Le but principal est d'enrichir les sources de données locales avec des données extraites à partir du web (également le Cloud) via les services de données. Cela permettrait de satisfaire les requêtes des utilisateurs tout en respectant leurs préférences en terme de coût d'exécution et de temps de réponse, en garantissant la qualité des résultats obtenus.

Dans une première partie de la thèse, nous décrivons le processus d'enrichissement des schémas des sources de données locales avec de nouveaux concepts nécessaires au traitement des requêtes de données des utilisateurs. Ce processus est basé sur deux algorithmes que nous avons développés. Le premier algorithme permet d'identifier l'information requise pour le traitement des requêtes mais qui n'existe pas dans la

source locale. Le deuxième algorithme permet d'enrichir le schéma de la source de données en rajoutant les éléments manquants déterminés par le premier algorithme.

Ensuite, nous démontrons comment le lac de services (Service lake) peut être exploité pour enrichir les sources de données locales. Dans ce cadre, nous proposons une nouvelle approche de composition de services orientée qualité afin d'identifier ceux qui sont pertinents et qui peuvent être utilisés pour obtenir l'information manquante. En particulier, nous avons adapté une approche LAV (Local As View) pour l'intégration de données, ainsi qu'une des solutions du problème de sac à dos pour la sélection des services pertinents. Ainsi, les données récupérées sont intégrées de manière transparente dans la base de données locale.

Afin de traiter l'hétérogénéité qui peut exister entre les données retournées par différents services et celles stockées dans les sources de données locales, nous proposons de définir des vues de services en terme des relations du schéma local. La première étape repose sur les résultats de matching calculés par COMA ++ entre la source de données et le service de données en question. Etant donné les mappings obtenus, on crée dans la deuxième étape un graphe pondéré. Par la suite, les top-k Steiner trees sont calculés.

Enfin, nous présentons MoDaaS qui est une plateforme de modélisation et de description des services de données, en particulier des services DaaS. Nous avons développé MoDaaS pour encourager les fournisseurs à adopter un modèle standard pour la modélisation des caractéristiques de leurs services , et ce selon une ontologie partagée, leur permettant ainsi de générer automatiquement des vues de service.

Mots clés: Intégration de données centrée sur l'utilisateur, Lacs de données, Enrichissement des schémas, Services de données, Données en tant que service (DaaS), Vues de service, Composition, Orchestration, Qualité de données, Qualité de service (QoS), Traitement des requêtes, Préférences d'utilisateur, Arbres Steiner, Nuage Informatique, Annotation sémantique, Ontologies de domaine , Ingénierie dirigée par les modèles (IDM), réutilisation et spécialisation.

Résumé Etendu

De nos jours, d'énormes volumes de données sont créés en continu et les utilisateurs s'attendent à ce que ceux-ci soient collectés, stockés et traités quasiment en temps réel. Ainsi, les lacs de données sont devenus une solution attractive par rapport aux entrepôts de données classiques coûteux et fastidieux (nécessitant une démarche ETL, Extract-Transform-Load), pour les entreprises qui souhaitent stocker leurs données. Malgré leurs volumes, les données stockées dans les lacs de données des entreprises sont souvent incomplètes voire non mises à jour vis-à-vis des besoins (les requêtes de données) des utilisateurs. Les sources de données locales ont donc besoin d'être enrichies. Par ailleurs, la diversité et l'expansion du nombre de sources d'information disponibles sur le web a rendu possible l'extraction des données en temps réel. Ainsi, afin de permettre d'accéder et de récupérer l'information de manière simple et interopérable, les sources de données sont de plus en plus intégrées dans les services Web. Il s'agit plus précisément des services de données, y compris les services DaaS (Data as a Service) du Cloud Computing. L'enrichissement manuel des sources locales implique plusieurs tâches fastidieuses telles que l'identification des services pertinents, l'extraction et l'intégration de données hétérogènes, la définition des mappings service-source, etc.

Plus précisément, nous considérons le scénario dans lequel un utilisateur (par exemple un employé) souhaite interroger un ensemble de données locale alors que cet ensemble de données ne contient pas des réponses complètes à toutes les requêtes de l'utilisateur. Cet ensemble de données peut être dans n'importe quel format, (par exemple, un fichier CSV, un document XML, une base de données relationnelle ou un graphique RDF). Généralement, les ensembles de données locaux doivent être complétés et enrichis avec des informations provenant de sources de données externes.

Considérons un ensemble de données contenant les tables relationnelles suivantes, où les attributs soulignés représentent des clés primaires. Nous supposons que les personnes, les auteurs et les livres sont identifiés de manière unique par leur identifiant. La clé étrangère authorID fait référence à une personne et l'auteur fait référence à l'authorID

de la table Author.

Schéma Relationnel
Person(personID, first name, last name, date of birth, country) Author(authorID, name, university, email, domain) Book(iD,title, author,topic)
Clés Etrangères
Table Author: authorID references personID of Person Table Book: author references authorID of Author

Considérons maintenant un utilisateur, qui est familier avec l'ensemble de données présenté ci-dessus, et qui est intéressé à émettre un ensemble de requêtes, présentées ci-dessous. L'utilisateur exprime ses requêtes de données en utilisant une syntaxe de langage de requête de type SQL dans laquelle les éléments requis par les requêtes mais qui ne sont pas réprésentés dans le schéma de la base de données sont pré-fixées par un point d'interrogation $'?'$.

Q_1 : `SELECT title, topic FROM Book`.

Q_2 : `SELECT ?iSBN, title FROM Book WHERE topic` $=$ $'$`Webservices`$'$.

Q_3 : `SELECT title, author, ?publisher FROM ?Publisher, Book WHERE ?Publisher .?name = Book.?publisher`.

L'exécution de Q_1 ne pose pas de problème. Par contre, les requêtes Q_2 et Q_3 ne peuvent pas être entièrement évaluées en se basant seulement sur la base de données locale introduite par l'utilisateur. La raison est que cet ensemble de données ne fournit pas tous les éléments nécessaires pour la l'évaluation de ces requêtes (i.e., les valeurs ISBN et éditeur n'existent dans aucun des tables de la base locale. De plus, la table Publisher n'est pas représentée dans le schéma). Différents services peuvent fournir une telle information. Également, un seul service ne garantit pas toujurs de répondre complètement aux requêtes des utilisateurs, ce qui rend indispensable la composition de plusieurs services web. Les services Web sont hétérogènes et généralement construits indépendamment du contexte dans lequel ils seront utilisés. Cela conduit à plusieurs problèmes de compatibilité (syntaxique, structurel ou sémantique). Ainis, cette grande disponibilité de services hétérogènes rend le processus de sélection et de composition des services une tâche non triviale. Heureusement, les données sont souvent associées

à certains critères(e.g., la qualité, le coût, la confidentialité et la sécurité, etc.) qui doivent être explicitement décrits et modélisés pour pouvoir sélectionner les meilleurs services satisfaisant les préférences des utilisateurs.

C'est dans cette optique que s'est situé notre projet de recherche, il consistait à proposer une nouvelle approche d'intégration de données centrée utilisateur. Le but principal est d'enrichir les sources de données locales avec des données extraites à partir du web (également le Cloud) via les services de données. Cela permettrait de satisfaire les requêtes des utilisateurs tout en respectant leurs préférences en terme de coût d'exécution et de temps de réponse, en garantissant la qualité des résultats obtenus.

Dans une première partie, nous décrivons le processus d'enrichissement des schémas des sources de données locales avec de nouveaux concepts nécessaires au traitement des requêtes de données des utilisateurs. Ce processus est basé sur deux algorithmes que nous avons développés. Le premier algorithme permet d'identifier l'information requise pour le traitement des requêtes mais qui n'existe pas dans la source locale. Le deuxième algorithme permet d'enrichir le schéma de la source de données en rajoutant les éléments manquants déterminés par le premier algorithme.

Ensuite, nous démontrons comment le lac de services (Service lake) peut être exploité pour enrichir les sources de données locales. Dans ce cadre, nous proposons une nouvelle approche de composition de services orientée qualité afin d'identifier ceux qui sont pertinents et qui peuvent être utilisés pour obtenir l'information manquante. En particulier, nous avons adapté une approche LAV (Local As View) pour l'intégration de données, ainsi qu'une des solutions du problème de sac à dos pour la sélection des services pertinents. Ainsi, les données récupérées sont intégrées de manière transparente dans la base de données locale.

Afin de traiter l'hétérogénéité qui peut exister entre les données retournées par différents services et celles stockées dans les sources de données locales, nous proposons de définir des vues de services en terme des relations du schéma local. La première étape repose sur les résultats de matching calculés par COMA ++ entre la source de données et le service de données en question. Etant donné les mappings obtenus, on crée dans la deuxième étape un graphe pondéré. Par la suite, les top-k Steiner trees sont calculés.

Enfin, nous présentons MoDaaS qui est une plateforme de modélisation et de description des services de données, en particulier des services DaaS. Nous avons développé MoDaaS pour encourager les fournisseurs à adopter un modèle standard pour la modélisation des caractéristiques de leurs services , et ce selon une ontologie partagée, leur

permettant ainsi de générer automatiquement des vues de service.

EuDaSL: Système d'Enrichissement automatique des Sources de Données Utilisateurs

Data Providing Service Lake

Nous introduisons dans cette thèse un nouveau paradigme que nous appelons le *Data Providing Service Lake* ou le Lac de Service de Données, par analogie avec le lac de données.

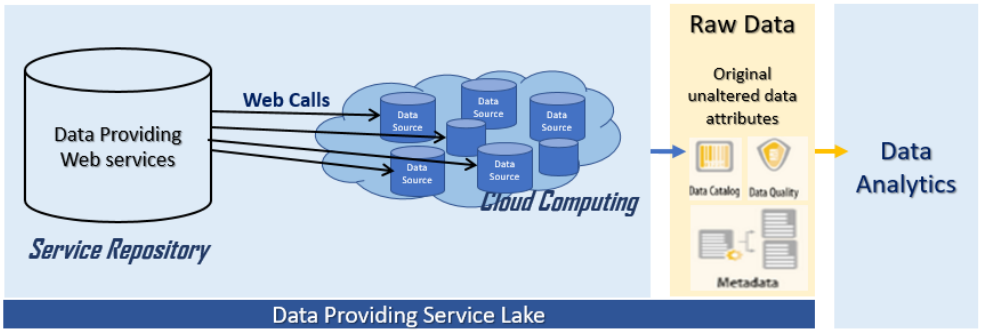


Figure 1 – Le Lac de Service

Un lac de service de données, ou le lac de service tout court, est un dépôt de stockage de services web de données hétérogènes fournissant un accès à la demande et en temps réel des informations de haute qualité. Les données renvoyées sont extraites à partir des sources web dans leur format natif et stockées dans les données brutes, telles quelles. L'idée principale des lacs de services est de tirer profit des services de données dans le lac, tandis que ces services rendent les données de sources web disponibles par le biais d'API encapsulées et de ne pas accorder trop d'attention à la création de schémas globales qui définissent des points d'intégration entre différents services web fournissant ainsi des ensembles de données hétérogènes. En conséquence, au lieu de placer les données récupérées à partir des différents sources hétérogènes dans un entrepôt de données spécialement conçu à cet effet, nous les déplaçons dans le lac, afin de les analyser ultérieurement. Cela permettrait de faciliter et d'enrichir dynamiquement les sources de données utilisateur en vue de répondre complètement et de satisfaire ses requêtes de données tout en éliminant les coûts initiaux et l'ingestion des données .

Architecture Globale

La figure 3.4 présente l'architecture globale de notre système d'intégration des données que nous appelons ***EuDaSL***(***E***nriching ***U***ser ***DA***ta sources in ***S***ervice ***L***akes). L'architecture de EuDaSL est composée principalement de quatre composants mettant

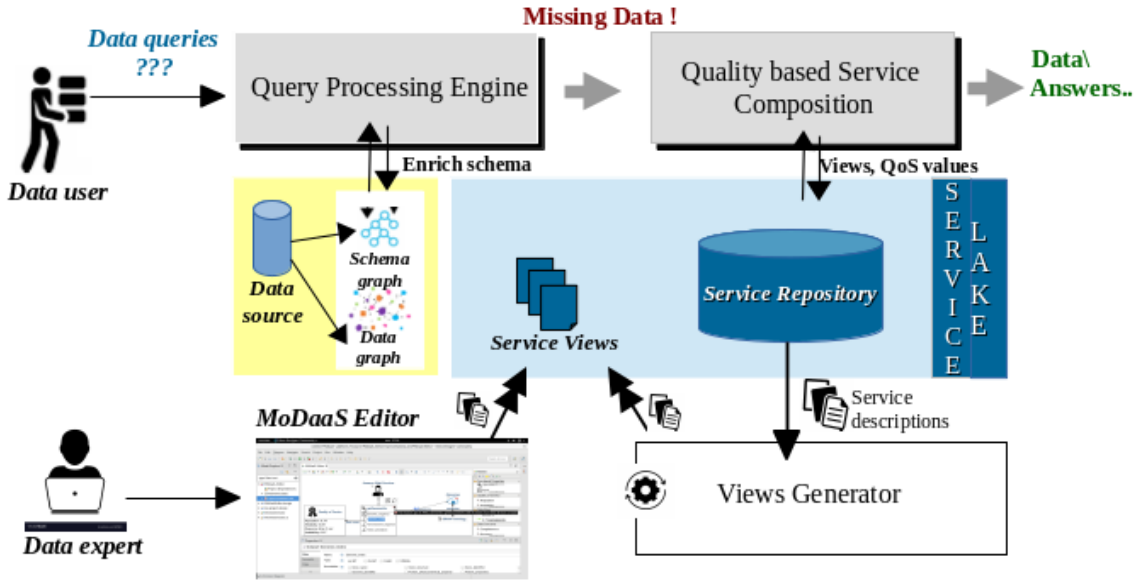


Figure 2 – Architecture Globale d'EuDaSL

en œuvre l'enrichissement des sources de données locales étant donné un ensemble de requêtes de données: le moteur de traitement de requêtes (Query Processing Engine), le module de composition de services de données (Quality based Service Composition), le générateur de vues de services (ViewsGenerator) et la plateforme de modélisation MoDaaS (MoDaaS Editor).

Processus d'Enrichissement

Suite à une requête de données, voire même un ensemble de requêtes, notre système doit déterminer la partie de la requête (les concepts et les relations) qui ne peut pas être satisfaite puisque l'information qu'elle cherche à trouver n'existe pas dans la source de données introduite par l'utilisateur. Une fois, tous les concepts et les relations manquants sont déterminés, le système les définit dans le schéma de la source de données. Ensuite, il procède à l'identification des services web de données candidates permettant d'extraire cette information, à l'extraction des données à partir des sources web

externes en invoquant les services web choisis et enfin il finit par l'intégration les données extraites dans la source de l'utilisateur. Notre système doit permettre entre autres la composition de différents services de données dans le cas où un seul service web ne permet pas d'avoir toute l'information recherchée.

La figure 3.5 illustre les différentes étapes du processus d'intégration proposé dans l'ordre suivant:

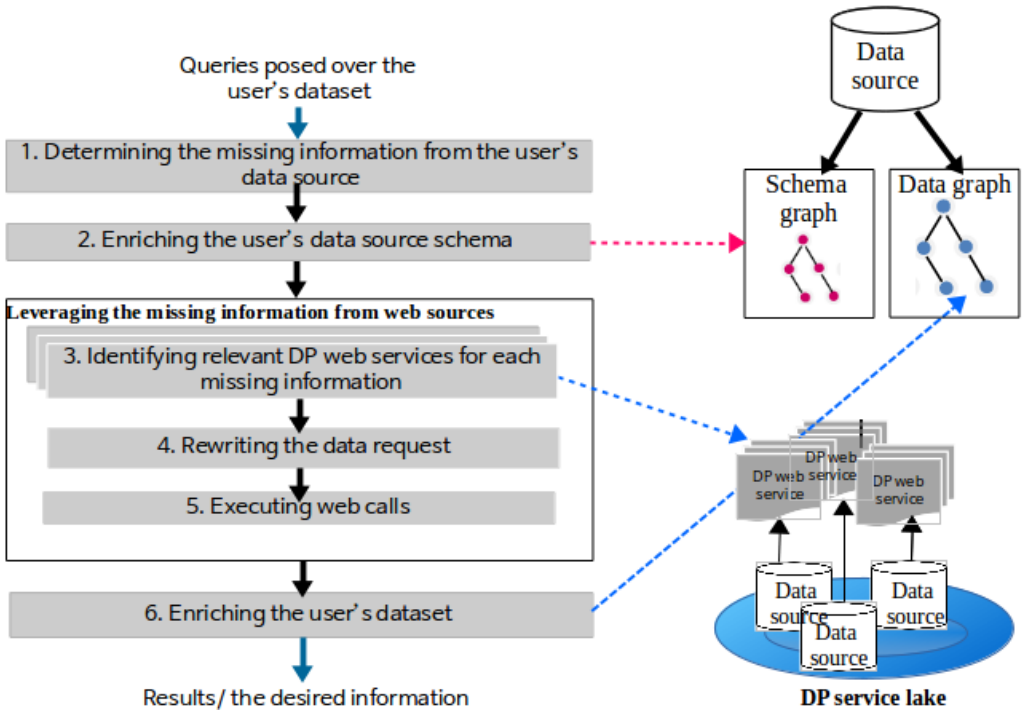


Figure 3 – Processus d'Enrichissement

1) la détermination de l'information recherchée par l'utilisateur mais qui ne figure pas dans la source de données introduite,

2) L'enrichissement du schéma de la source de données utilisateur, en introduisant les concepts qui y manquent,

3) L'identification des services de données pertinents permettant de fournir l'information manquante,

4) La reformulation des requêtes utilisateur pour inclure des appels web aux services de données sélectionnés,

5) L'exécution des appels web afin d'extraire les données, indispensables pour satisfaire les requêtes des utilisateurs,

6) Et finalement l'intégration des données extraites via les services de données dans la source de données locale.

Toutes ces étapes présentent des tâches fastidieuses ainsi que leur réalisation manuelle nécessite beaucoup de temps, d'effort et d'expertise. Notre approche permet de rentabiliser les efforts requis par ces tâches et réduire la complexité de programmation tout en garantissant un coût d'exécution minimal et une meilleure qualité des résultats obtenus.

Identification des Données Manquantes (Missing Data)

Dans notre approche, la source de données est représentée en un 'graphe de schéma' et un 'graphe de données'.

Le graphe de schéma est un graphe orienté qui représente le schéma de la source de données (cf. Figure 3.1), alors que graphe de données représente l'ensemble des données/instances qui figurent dans la source tout en respectant la structure représentée dans graphe de schéma. Cette distinction est nécessaire pour assurer une meilleure performance des algorithmes développés.

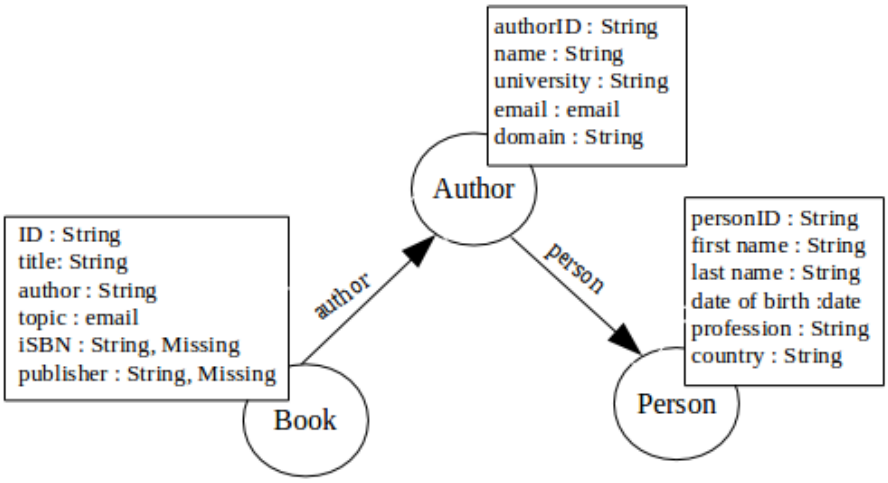


Figure 4 – Un Exemple d'un Graphe de Schéma

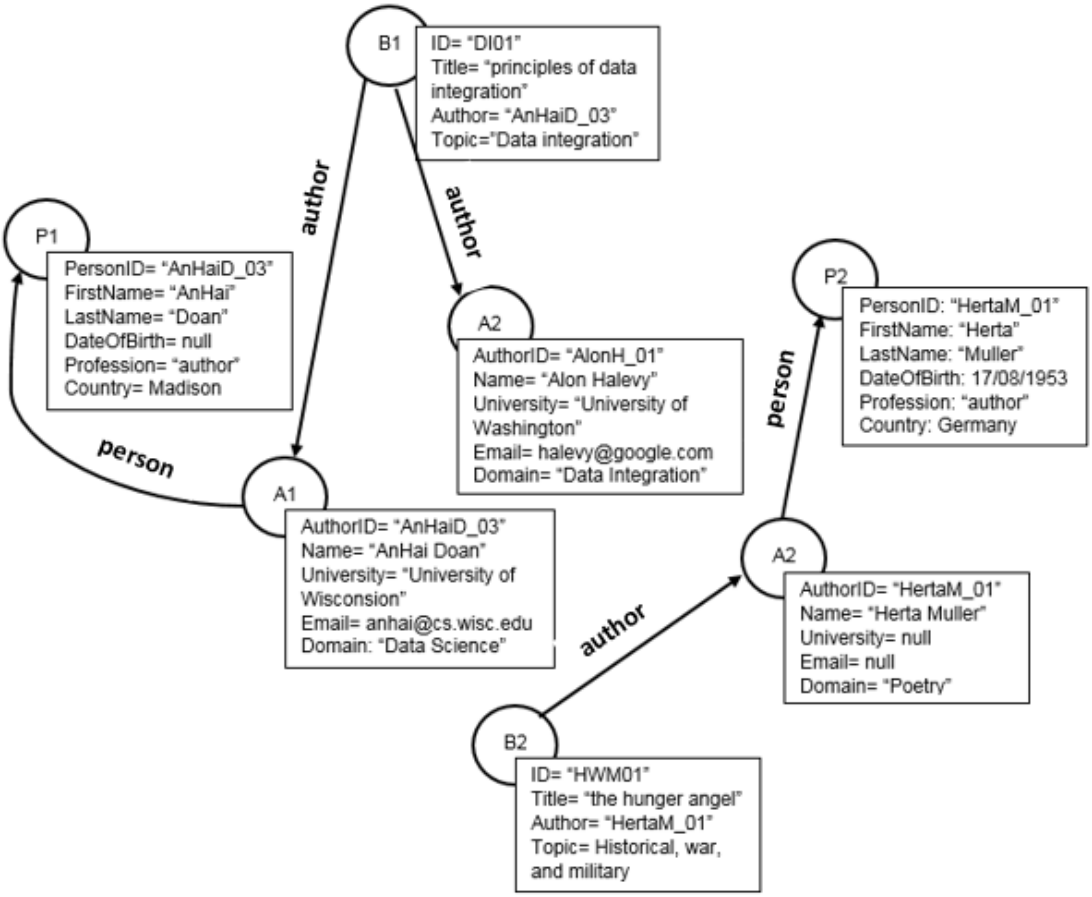


Figure 5 – Un Fragment du Graphe de Données

- Un premier algorithme (Algorithme 6) permet de parcourir l'ensemble des requêtes des utilisateurs et le schéma de la source de données introduite pour déterminer les concepts et les attributs qui sont demandés par l'utilisateur mais ne sont pas représentés dans le 'schema graph'.

Algorithme 1 : Searching for missing information

Require : Q_D= Q_1, Q_2, ..., Q_n is a data request, G_S= (V,E) is a schema graph
Ensure : MissElts= MissConcepts, MissAttributes, MissRelations
$MissConcepts \leftarrow \emptyset$, $MissAttributes \leftarrow \emptyset$, $MissRelations \leftarrow \emptyset$
ForEach Q_i *in* Q_D **do**
  if Q_i.*concepts involves only one concept c* **then**
    if *there is a node* $v \in V$ *that corresponds to c* **then**
      ForEach Q_i *in* Q_D **do**
        attribute att $\in Q_i.attributes$ that does not belong to v.attributes
      add (att, c, 'certain') to MissAttributes
    end

```
7
8
9       else if c was already defined in MissConcepts then
10        foreach attribute att ∈ Q_i.attributes do
11          if (att, concepts, certitude) was defined in MissAttributes such as
              c ∈ concepts and certitude='uncertain' then
12            replace (att, concepts, uncertain) by (att, c, 'certain')
13          else if att was not defined in MissAttributes then
14            add (att, c, 'certain') to MissAttributes
15      else add c to MissConcepts and all attributes in Q_i.attributes to
          MissAttributes;
16    else foreach concept c ∈ Q_i.concepts that does not have any representative
        node v ∈ V do
17      add c to MissConcepts
18    foreach condition cond ∈ Q_i.conditions do
19      get related attribute-concept pairs < att, c > in cond
20      foreach < att, c > such as (c is not missing from G_S and att is missing)
          or (c was defined as a missing concept however att was not defined as a
          missing attribute related to c) do
21        add (att,c,'certain') to MissAttributes
22      foreach linked concepts c1 and c2 in cond that are not related in G_S by
          an edge e ∈ E do
23        define a new edge e= (c1, c2, att1) in MissRelations where c1
            represents the outgoing node, c2 is the incoming node and att1 is
            the label of this edge
24    foreach a in Q_i.attributes that does not belong to any c. attributes and was
        not defined as a missing attribute related to c such as c ∈ Q_i.concepts do
25      concepts ← ∅
26      foreach c in Q_i.concepts do
27        compute the relatedness score("a", "c")
28        if c has a relatedness score higher than 0.5 then
29          add c to concepts
30      add(a, concepts, 'uncertain') to MissAttributes
31    ;
```

- Un deuxième algorithme (Algorithme 7) permet d'enrichir et d'augmenter le schéma de la source locale (schema graph) en définissant l'ensemble des 'missing concepts/ attributs' et des 'missing relations' qui ont été déterminé par le premier algorithme.

Algorithme 2 : Enriching Schema Graph

Require : $G_S = (V, E)$, MissElts= MissConcepts, MissAttributes, MissRelations

Ensure : G_S (the enriched Schema Graph)

```
foreach c ∈ MissConcepts do
    add a new node named c with the label 'M' to V
foreach att ∈ MissAttributes do
    foreach c ∈ attribute.concepts do
        define a new attribute att additionally to initial attributes defined within
         the concept c, having 'String' as a type and 'Missing' as a state
foreach rel ∈ MissRelations do
    add a new edge e to E outgoing from rel.OutNode, incoming to rel.InNode
     and labeled with rel.label
```

La figure 3.6 représente un exemple d'enrichissement du Schema Graph présenté dans la Figure 3.1. Les noeuds, les attributs et les arrêtes en gris représentent respectivement les nouveaux concepts, les nouveaux attributs et les relations qui sont indispensables pour le traitement des requêtes de données posées par l'utilisateur mais qui ne sont pas définis dans le schema graph initial de la base de données introduite.

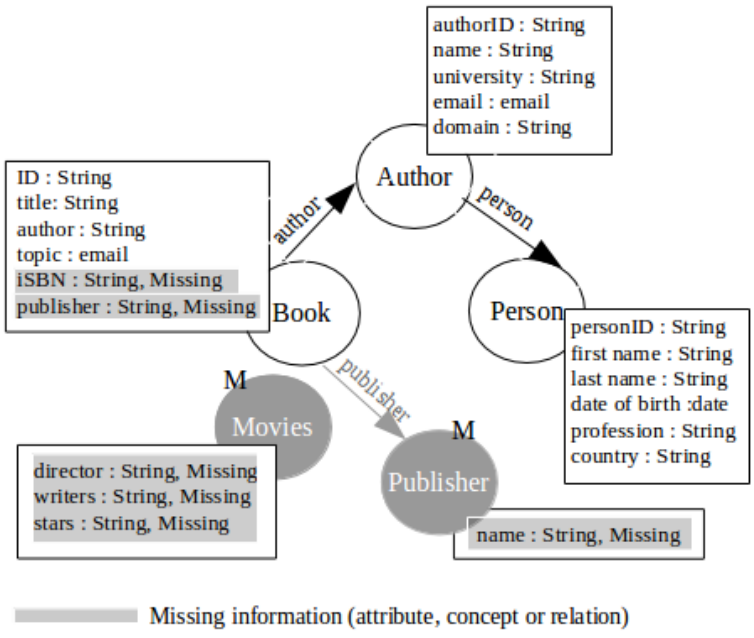


Figure 6 – Un Exemple d'Enrichissement du Schema Graph

Sélection & Composition des Services de Données Pertinents en Vue de l'enrichissement des ensembles de Données Utilisateur

Dans cette partie, nous décrivons notre approche de sélection et de composition des services web de données. Une première étape consiste à déterminer les services de données candidats permettant d'obtenir tout ou une partie des données manquants de la source de données locale. Une deuxième étape permet de construire les différentes compositions possibles et les plans de requête correspondants permettant ainsi de retrouver toute l'information désirée. Quant à la dernière étape consiste à évaluer la qualité des compositions en termes de qualité de données (QD) et de la qualité de service (QoS) et ainsi sélectionner ceux permettant de satisfaire les requêtes de données, y compris les contraintes des utilisateurs (e.g., temps de réponse, coût d'exécution).

Composition des Services de Données Pertinents

Étant donné une requête de données Q et un ensemble de services de données représentés par leurs vues $V = v_1, v_2, ..., v_i$, l'algorithme 8 permet d'identifier toutes les vues de services candidats contenant tout ou une partie des conjonctions constituant la requête Q.

Tout d'abord, l'algorithme commence par vérifier que l'information présente dans la source de données de l'utilisateur est utilisée en premier lieu pour répondre à la requête Q. Si certaines données manquent de la base de données locale, l'algorithme procède pour identifier l'ensemble des vues des services de données dans le Service Lake permettant de renvoyer ces données manquantes. Ensuite, l'algorithme crée toute les combinaisons possibles des différents services sélectionnés. Une combinaison est censé renvoyer tout l'ensemble des données manquantes. Dans ce qui suit, nous montrons comment créer un plan d'exécution de requête à partir d'une composition de services.

Création des Plans d'Exécution (Query Plans)

La création des plans d'exécution (réalisée par l'algorithme 9) est un processus de raffinement permettant de passer d'une combinaison de services, chacun renvoyant une partie de l'information recherchée par la requête de données, vers une description

Algorithme 3 : Identify Relevant Service Views

Require : $\mathtt{Q}(\overline{\mathtt{X}}) \rightarrow \mathtt{q}_1(\overline{\mathtt{X}}_1), \ldots, \mathtt{q}_m(\overline{\mathtt{X}}_m), \mathtt{C}_I$ is a conjunctive query
$\mathtt{SV}$ a set of service views
Ensure : $\{\mathtt{RelevantViews}_1, \ldots, \mathtt{RelevantViews}_m\}$

1 **for** *every conjunct* $\mathtt{q}_i$ *in* $\mathtt{Q}$ **do**
2 $\mathtt{RelevantViews}_i \leftarrow \emptyset$
3 **if** $\mathtt{q}_i(\overline{\mathtt{X}}_i)$ *is not missing* **then**
4 there exists a relation $\mathtt{R}(\overline{\mathtt{Y}})$ that corresponds to $\mathtt{q}_i(\overline{\mathtt{X}}_i)$
5 let $\mathcal{M}$ be the mapping defined on the variables of $\mathtt{R}(\overline{\mathtt{Y}})$ as follows: **if** $\mathtt{Y}$ *is the j'th variable of* $\mathtt{R}$ **then**
6 $\mathcal{M}(\mathtt{Y}) \leftarrow \mathtt{X}_j$ where $\mathtt{X}_j$ is the j'th variable in $\overline{\mathtt{X}}_i$
7 add $\mathcal{M}(\mathtt{R})$ to $\mathtt{RelevantViews}_i$
8 **else**
9 **for** *every conjunct* $\mathtt{u}(\overline{\mathtt{Y}})$ *in a body of a service view* $\mathtt{sv}$ *in* $\mathtt{SV}$ **do**
10 **if** $\mathtt{q}_i = \mathtt{u}$ **then**
11 let $\mathcal{M}$ be the mapping defined on the variables of $\mathtt{sv}$ as follows: **if** $\mathtt{Y}$ *is the j'th variable of* $\mathtt{u}$ *and* $\mathtt{Y}$ *is not existential* **then**
12 $\mathcal{M}(\mathtt{Y}) \leftarrow \mathtt{X}_j$ where $\mathtt{X}_j$ is the j'th variable in $\overline{\mathtt{X}}_i$
13 **else**
14 $\mathcal{M}(\mathtt{Y})$ is a new variable that does not appear in $\mathtt{Q}$ or $\mathtt{sv}$
15 add $\mathcal{M}(\mathtt{sv})$ to $\mathtt{RelevantViews}_i$

exécutable où tous les services composants peuvent être exécutés. Un exemple d'un plan d'exécution est montré dans la figure 4.2, où chaque nœud correspond à l'invocation d'une opération d'un service de la composition, et les arcs servent à exprimer l'ordre d'enchaînement des appels web.

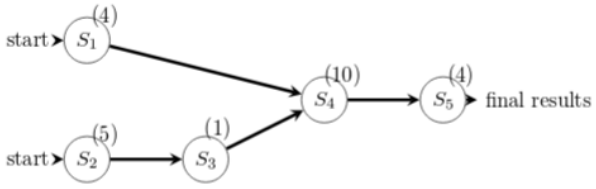


Figure 7 – Un Exemple de Plan d'Exécution P_1

L'exécution du Plan de requête P_1 commence par l'invocation des services S_1 et S_2. Une fois le service S_2 retourne des valeurs (des données), le service S_3 sera être exécuté autant de fois que le nombre de données renvoyées par S_2. De même, l'exécution du service S_4 ne peut être déclenchée qu'après la fin de l'éxécution des services S_1 et S_3,

suivi par l'appel du service S_5 permettant de renvoyer l'information recherchée par la requête Q. Algorithme 9 consiste à définir le plan d'exécution rendant une composition

Algorithme 4 : Create Executable Query Plans

Require : a conjunctive plan $\mathtt{Q}$‘ corresponding to a possible combination of relevant views $(\mathtt{R_1}, ..., \mathtt{R_l}, \mathtt{V_1}, ..., \mathtt{V_m})$
I/O parameters of a data service V_i is $(\mathtt{in_i}, \mathtt{out_i})$

Ensure : a query plan $\mathtt{P}$

1 $\mathtt{BindAvail_0}$ = The set of variables in Q‘ bound by values in the query and those returned by local relations $R_i, i \in [1..l]$
2 Q_{out} = The head variables of Q‘ (the same head variables of Q).
3 $j = 2$
4 **while** *there exists also service views that were not chosen earlier* **do**
5 **foreach** V_i *in* Q **do**
6 **if** V_i *was not chosen earlier and the parameters in* in_i *are in* $\mathtt{BindAvail_{i\_1}}$ **then**
7 V_i is the j‘th service in the ordering
8 $BindAvail_i = BindAvail_{i\_1} \cup out_i$
9 $j \leftarrow j + 1$
10 **if** $Q_{out} \not\subseteq BindAvail_m$ **then**
11 plan is not executable

de service exécutable. Cependant, si aucun plan possible n'est trouvé, la composition ne sera pas être considérée dans la suite du processus de sélection.

Sélection des Services basée sur la Qualité

Cette étape nécessite une première étape d'évaluation de la qualité des différentes compositions de services exécutables, identifiées dans l'étape précédente, suivie par la sélection de la meilleure combinaison des compositions qui maximise la qualité des données retournées tout en respectant les préférences des utilisateurs.

On associe à chaque composition i un score de qualité q_i et un coût c_i. Le score de qualité est défini comme une agrégation additive de différents critères de qualité de service (QoS) tels que la disponibilité, la réputation, le temps de réponse et le coût d'exécution des services composants par sommation pondérée. Pour ce faire, nous utilisons la méthode SAW (Simple Additive Weighting), une méthode d'aide à la décision multi-critères .

Le problème est donc de choisir un ensemble de compositions de services de telle manière que l'on ne dépasse pas le budget maximal de l'utilisateur tout en maximisant la somme des qualités associées aux compositions sélectionnées. Ce problème est reformulé en tant qu'un problème de sac à dos (0-1 knapsack problem) comme suivant:

Étant donné plusieurs compositions de services possédant chacune un score de qualité et un coût d'exécution et étant donné un coût d'exécution maximale C spécifié par l'utilisateur pour le sac, quels compositions faut-il mettre dans le sac de manière à maximiser le score de qualité totale sans dépasser le coût d'exécution maximal autorisé pour le sac ?

$$\texttt{Max} \sum_{\texttt{i} \in \texttt{S}} \texttt{q}_\texttt{i} \texttt{ such that } \sum_{\texttt{i} \in \texttt{S}} \texttt{c}_\texttt{i} \leq \texttt{C}$$

L'algorithme 10 permet d'identifier la combinaison optimale des services permettant de renvoyer le maximum de réponses pour un coût total ne dépassant pas le budget maximal de l'utilisateur. Ainsi, les données récupérées sont intégrées de manière transparente dans la base de données locale.

Spécification des Vues de Services de Données dans le Service Lake

Les services Web sont hétérogènes et généralement construits indépendamment du contexte dans lequel ils seront utilisés. Cela peut entraîner plusieurs problèmes de compatibilité (syntaxique, structurel ou sémantique) et rend la sélection et la composition des services une tâche fastidieuse. Afin de traiter l'hétérogénéité qui peut exister entre les données retournées par différents services et celles stockées dans les sources de données locales, nous proposons de définir des vues de services en termes des relations du schéma local.

Vue de Service: est une reformulation de la description du service de données (ses paramètres d'entrée/ sortie) en fonction des concepts de la source de données introduite par l'utilisateur.

La première étape repose sur les résultats de matching, ou aussi appelés mappings,

Algorithme 5 : Select Plans

Require : n: the number of plans, which are labeled 1 to n
q_a: an array containing the values of the qualities of the plans
c_a: an array containing the costs of the plans
C : the maximum cost allowed

Ensure : Q, an array of the best qualities obtained considering a given plan under a cost constraint

```
1  for (c = 0 to C) do
2      Q[0,c] = 0
3  for (i = 1 to n) do
4      for (c = 0 to C) do
5          if ((c_a[i] ≤ c) and (q_a[i] + Q[i-1,c - c_a[i]] > Q[i-1,c])) then
6              Q[i,c] = q_a[i] + Q[i-1,c - c_a[i]]
7              keep[i,c] = 1
8          else
9              Q[i,c] = Q[i-1, c]
10             keep = [i,c] = 0
11 T = C for (i = n downto 1) do
12     if (keep[i,T] = 1) then
13         Print i
14         T = T - c_a[i]
15 Return Q[n,C]
```

calculés par COMA ++ entre la source de données et le service de données en question. Le processus de matching permet de définir la correspondance entre deux modèles de données, dans notre cas décrire le lien entre le schéma global de la base de données locale et les schémas des services de données.

Etant donné les mappings obtenus, on crée dans la deuxième étape un graphe pondéré, tels que les poids sur les noeuds représentent les scores de matching correspondants. Par la suite, les top-k Steiner trees sont calculés.

Problème de l'arbre de Steiner: étant donné le schema graph G_S et les mappings calculés par COMA++, l'objectif est de trouver les top-k arbres de Steiner reliant tous les sommets de correspondance (contenant au minimum une correspondance d'un paramètre de service) dans le graphe, classés par leurs scores de matching. Ainsi, les arbres trouvés sont considérés les vues du service de données en question.

La pertinence d'un arbre de solutions T peut être mesurée à l'aide des scores des noeuds comme suit:

$$\texttt{Rel}(\texttt{T}) = \sum_{\texttt{v} \in \texttt{V}(\texttt{T})} \texttt{score}(\texttt{v}) \tag{1}$$

Nous avons implémenté un algorithme pour calculer les top-k arbres de Steiner interconnectant différents sommets V_i, chacun représente un ou plusieurs paramètres de service. L'objectif étant de maximiser le score de correspondance (le matching score), cet arbre avec le meilleur score peut représenter la meilleure reformulation de la vue de service en question.

Modélisation & Description des Services DaaS

Dans ce dernier travail, nous nous sommes intéressés à développer une approche dirigée par les modèles pour la description et la modélisation des services de données, et ainsi pour l'assistance à la sélection et la composition automatique des services DaaS. Cette approche permet à la fois de simplifier et semi-automatiser la spécification des vues de services DaaS. L'idée principale derrière est de décrire les services en tant que des vues RDF paramétrés en se basant sur une ontologie de médiation afin de capturer leur sémantique de manière déclarative et les vues définies sont ensuite utilisées pour la description des différents services et elles sont exploitées dans la composition automatique des services.

Dans ce qui suit,nous allons présenter une vue globale de la solution proposée.

MoDaaS : Vers un Modèle Générique de Description et Modélisation des Services de Données (DaaS)

Afin de mettre en oeuvre l'approche retenue, nous avons choisi d'utiliser des standards et des outils issus de l'ingénierie dirigée par les modèles (IDM) afin d'assurer la portabilité de la solution. En fait, l'IDM présente est une solution prometteuse quand il s'agit de traiter les problématiques liées à la portabilité et à la gestion de l'hétérogénéité. Nous avons adopté en particulier le principe de semi-automatisation du processus de transformation de la méthodologie MDA (Model Driven Architecture). L'approche MDA permet la séparation des spécifications fonctionnelles d'un système des détails de son implémentation sur une plateforme donnée, ce qui nous a permis de définir différents services DaaS implémentés par différents fournisseurs et générer les

vues RDF corresponds en utilisant les concepts d'une même ontologie standard.

Nous commençons dans un premier temps par la description de l'architecture globale. Ensuite, nous décrivons les différents modules plus en détails.

Architecture Globale

Au cours de ce projet, deux outils ont été développés:

- **MoDaaS Editor :** un éditeur graphique permettant aux fournisseurs DaaS de définir et de décrire leurs services de données. Cet éditeur se repose sur le métamodèle "DaaSMetaModel". Ce dernier regroupe les concepts nécessaires pour la définition d'un modèle DaaS en utilisant des classes personnalisables et extensibles.
- **Views Generator :** un outil génératif permettant de générer automatiquement les vues RDF des services de données à partir des modèles DaaS fournis en entrée.

La figure 6.2 présente les différents outils qui composent notre framework MoDaaS et les relations entre eux.

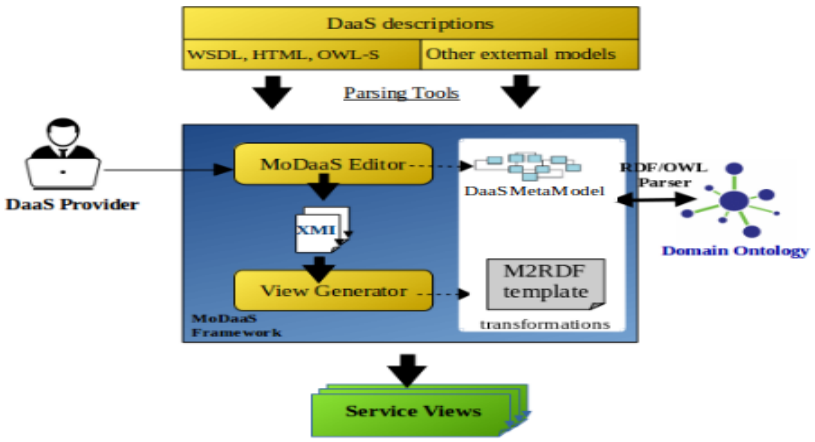


Figure 8 – Overall Architecture of the MoDaaS Framework

DaaSMetaModel:

La figure 6.3 présente le méta-modèle au coeur de notre plateforme MoDaaS. DaaSMetaModel regroupe les différents concepts et propriétés des services de données du

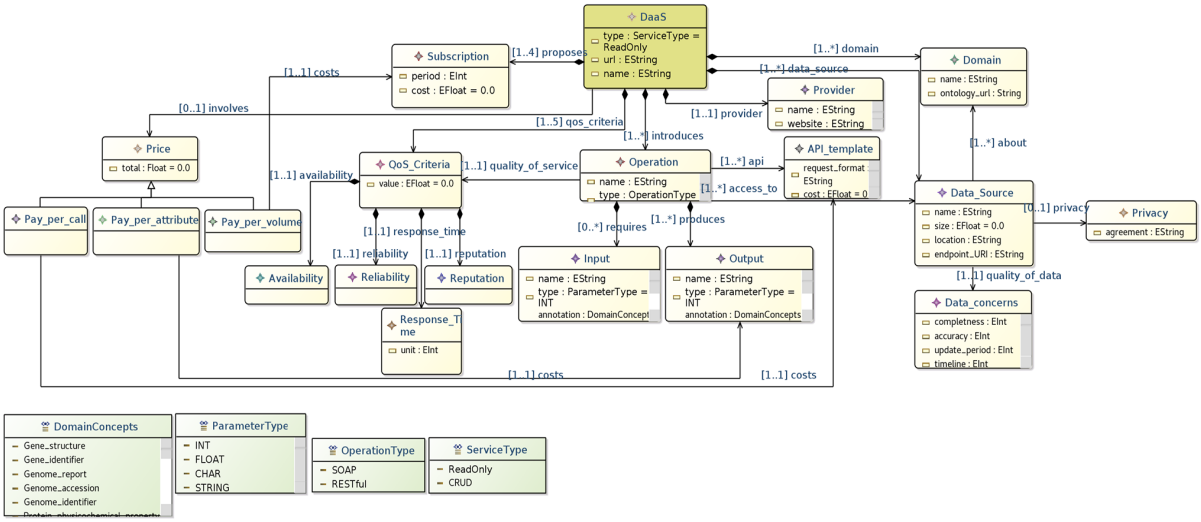


Figure 9 – DaaSMetaModel

nuage informatique DaaS. Les classes présentées dans ce méta-modèle permettent de décrire le service DaaS, ses paramètres d'entrée et de sortie, ainsi que ses propriétés en termes de qualité de service (QoS) et de la qualité de données (QD)

MoDaaSEditor:

L'éditeur graphique MoDaaSEditor permet aux utilisateurs et aux fournisseurs DaaS de créer et de décrire leurs services de données à partir des différents éléments présentés dans la palette correspondante (cf. Figure 6.4).

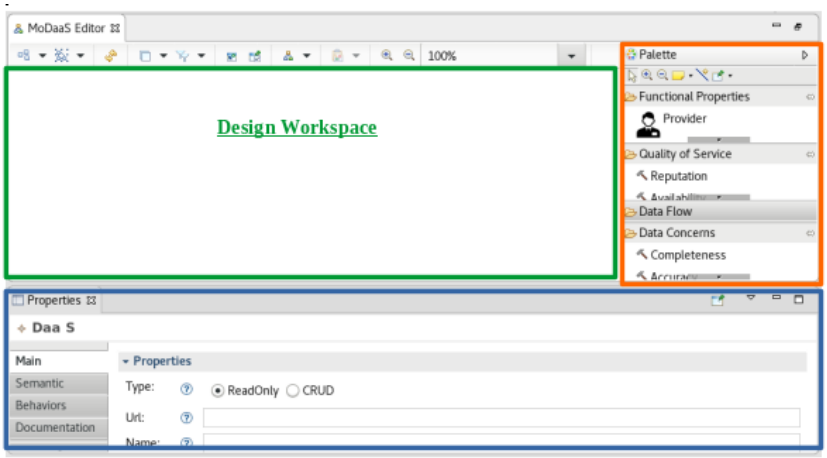


Figure 10 – MoDaaSEditor

ViewsGenerator: Génération Automatique des Vues RDF

Le générateur ViewsGenerator permet de générer la vue de service (RDF) à partir du modèle DaaS créé. Il prend en entrée le fichier DaaS.daas et retourne un fichier view.rdf contenant l'ensemble des paramètres d'entrée et de sortie exprimés en fonction des concepts de l'ontologie choisie.

Implémentaion & Validation

Afin de valider notre approche, nous avons réalisé trois différents cas d'étude concrets. Il s'agit de la modélisation et la génération des vues RDF de trois différents services DaaS en fonction des concepts de l'ontologie EDAM.

Implémentation

Après avoir étudié différents outils de modélisation et de transformation de modèles en texte, nous nous sommes amenés à choisir :

- **EMF (Eclipse Modeling Framework):** le plugin de base de la métamodélisation sous Eclipse.
- **Ecore:** utilisé pour la définition du méta-modèle DaaSMetaModel.
- **Sirius:** utilisé pour la définition et la mise en oeuvre d'une syntaxe concrète graphique pour notre DSML (DaaSEditor).
- **Acceleo:** comme outil de transformation utilisé pour la génération automatique des vues de services.

Dans ce qui suit, nous allons présenter des imprimes écran illustrant le processus de modélisation et la génération des vues de services DaaS en utilisant les outils développés.

Description & Modélisation des Services DaaS

Dans cette partie, nous allons voir comment décrire le service DaaS ***getGenomeInfo*** proposé par ***Amazon Web Services*** et générer sa vue RDF en utilisant notre plateforme MoDaaS.

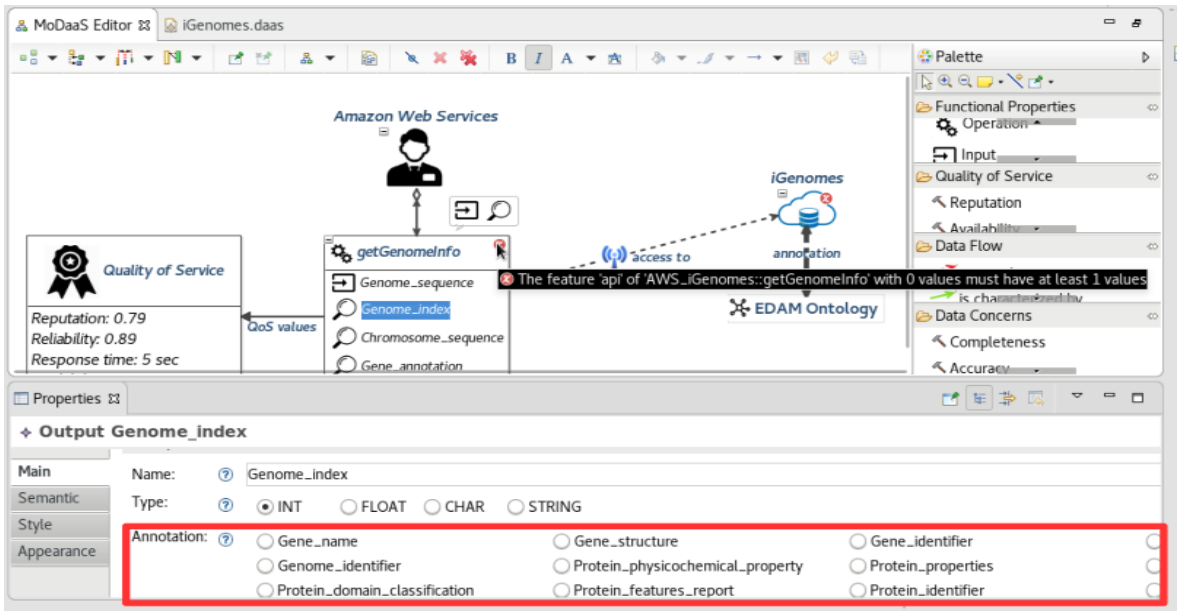


Figure 11 – Modélisation du service getGenomeInfo et Annotation sémantique selon l'ontologie EDAM

Le processus débute par la création d'un modèle DaaS décrivant le service de données *getGenomeInfo* à partir des différents éléments de la palette présentée par notre éditeur graphique MoDaaSEditor. Ces éléments présentent les différents paramètres à savoir les paramètres d'entrée requis pour l'exécution du service et les paramètres de sortie recherchés par la requête (de l'utilisateur). Ils permettent aussi de définir la qualité de données fournies en sortie, la qualité de service (QoS) en tant que la réputation du fournisseur, la disponibilité du service, etc.

La définition des caractéristiques de chaque composant du modèle créé se fait à partir de la vue "Properties". Cette vue présente l'ensemble des propriétés à spécifier pour chaque élément. En effet, il faut tout d'abord sélectionner le composant et par la suite remplir les champs apparus dans "Properties". Suite à La création d'un nouveau modèle DaaS, un nouveau fichier *.daas est créé pour représenter convenablement les différentes propriétés et caractéristiques du service *getGenomeInfo* par ce modèle au format XMI. La figure 6.8 présente le contenu du fichier iGenomes.daaS.

De plus, l'utilisateur peut à tout moment effectuer une validation du modèle créé grâce à MoDaaSEditor qui fait appel à son tour au méta-modèle DaaSMetaModel pour vérifier la validité du modèle DaaS créé. La validation est lancée depuis MoDaaSEditor.

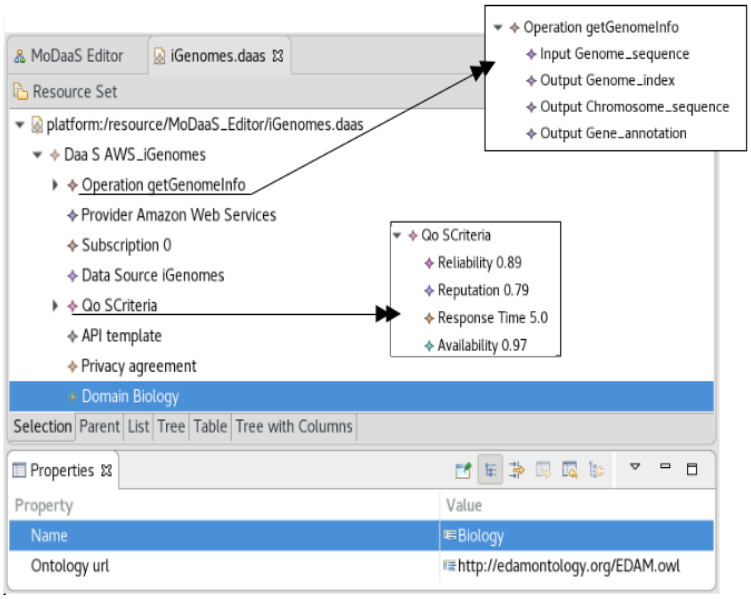


Figure 12 – Modèle de description du service getGenomeInfo (*.daas) au format XMI

Génération des Vues RDF de Services DaaS Après avoir créé et validé le modèle DaaS, l'outil ViewsGenerator permet de générer automatiquement la vue RDF du service à partir de ce modèle selon une ontologie de domaine, à l'état actuel seulement l'ontologie EDAM est supportée par notre plateforme de modélisation. Ainsi, pour lancer la génération de la vue RDF, il suffit de cliquer sur le fichier iGenomes.daas et puis "Generate View". Le listing 6.1 présente le code généré par ViewsGenerator à partir du modèle DaaS créé.

```
1 getGenomeInfo($genome_sequence, ?genome_index, ?Chromosome_sequence, ?gene_annotation
    '): ?<http://edamontology.org/data_3210> <http://www.w3.org/2000/01/rdf-schema#
    label> "Genome index"
2 ?<http://edamontology.org/data_0919> <http://www.w3.org/2000/01/rdf-schema#label> "
    Chromosome report"
3 ?<http://edamontology.org/data_0916> <http://www.w3.org/2000/01/rdf-schema#label> "
    Gene report"
```

Listing 1 – Vue RDF du service *getGenomeInfo*

De cette façon, nous décrivons les services de données fournis par les fournisseurs de DaaS en tant que des vues selon une ontologie de domaine afin de traiter l'hétérogénéité des schémas. Plus précisément, les paramètres de chaque service DaaS sont matchés avec les concepts qui les réprésentent dans l'ontologie de domaine choisie. Ces concepts sont utilisés pour définir les vues RDF des services. Les vues de service générées sont ensuite stockées dans un référentiel et peuvent donc être réutilisées automatiquement par les outils de sélection et de composition automatiques.

Contents

List of Figures

List of Tables

Liste des symboles

DaaS ..... Data as a Service

DP service ..... Data Providing service

ETL ..... Extract, Transform and Load

iPaaS ..... integration Platform as a Service

JSON ..... JavaScript Object Notation

ODL ..... Object Definition Language

OWL-S ..... OWL for Services

OWL ..... Web Ontology Language

PRV ..... Parametrized RDF View

QoD ..... Quality of Data

QoS ..... Quality of Service

RDF ..... Resource Description Framework

REST ..... REpresentational State Transfer

SAWSDL ..... Semantic Annotations for WSDL

SLA ..... Service Level Agreement

SOA ..... Service Oriented Architecture

SOAP ..... Simple Object Access Protocol

SOC ..... Service Oriented computing

SQL ..... Structured Query Language

SWS ..... Semantic Web Services

UDDI ..... Universal Description Discovery and Integration

UML ..... Unified Modelling Language

URL ..... Uniform Resource Locator

WSDL ..... Web Services Description Language

WWW ..... World Wide Web

XML ..... Extensible Markup Language

XSLT ..... Extensible Stylesheet Language Transformations

Chapter 1

Introduction

Contents

This chapter introduces the scope of the work achieved throughout this PhD thesis. This thesis is accomplished at the *LAMSADE* Laboratory of the university of Paris Dauphine in France and the *RIADI* laboratory of the university of Manouba in Tunisia. It is part of the *PHC-Utique* research project n°**15G1413**, entitled "***Data Integration in Cloud Environments***". This collaboration through the PHC-Utique project guided the objectives of the thesis, its orientations and its products. In particular, our task consisted in proposing a new approach for the on-the-fly enrichment of local data sources with new information coming from external data sources. As such, we present in this chapter the context of our work, its underlying challenges and our main contributions.

1.1 Research Context & Motivation

With the increasing and continuous production of large amounts of data coming from a variety of sources, existing data processing technologies are no more suitable to cope with it. Doing so, we observe a shift in the way companies are managing their data.

Big data platforms and analytic architectures have recently witnessed a wide adoption by companies and businesses world-wide to transform their increasingly growing data into actionable knowledge. Indeed, big data frameworks help in storing, analyzing and processing such massive amounts of data, and thus identifying new opportunities through enabling faster and better decision making, more efficient operators and higher profits, and also lead to more satisfied customers by gauging their needs. In particular, managing "integrated" data assures more confidence in decision-making and provides superior insights, and thus the need for faster and smarter data integration capabilities is growing.

Generally speaking, data integration combines data originating from a variety of different and disparate sources and software formats, and then provides users with a unified view of the accumulated data. The main aim is to make data more comprehensively available, and to increase the value of existing data by allowing previously complex data queries to be posed upon it. The proliferation of disparate data sources, heterogeneous types, and web stores is increasing the challenge of combining data into meaningful and valuable information. Thereafter, the process of integrating huge and heterogeneous data sets is becoming increasingly complex and challenging considering the current "data explosion". The complexity emphasizes on the levels of data interoperability, data storage, structure and the levels at which the data can be integrated and operated as a single entity. Furthermore, collecting and maintaining large data sets is costly. Therefore, organizations tend to use the cloud for storing their data. Consequently, the biggest challenge is incorporating all relevant data across an ever-increasing number of databases on the web and possibly on the cloud with on-premises databases that were designed and created independently the one from one another. Thus, bringing data together is not a trivial exercise.

Specifically, we consider the scenario in which a user (e.g., a company employee) wants to query a local dataset while this dataset may not be sufficient by itself to provide answers for all the user' queries. This dataset can be in any format, (e.g., a CSV file, an XML document, a relational database or an RDF graph) and may contain for example information about prospective clients of companies. Often, local datasets need to be augmented and enriched with information coming from external data sources. Our focus in this thesis is to fill the informational gaps in the local databases with information coming from external data sources whenever the stored data is not sufficient for answering a data query.

Let consider a dataset containing the following relational tables, where underlined attributes represent primary keys. We assume that persons, authors and books are

uniquely identified by their id. The foreign key *authorID* references a *person* and *author* references *authorID* of the table *Author*.

Relation schema
Person(personID, first name, last name, date of birth, country) Author(authorID, name, university, email, domain) Book(iD,title, author,topic)
Foreign Keys
Table Author: authorID references personID of Person Table Book: author references authorID of Author

Consider now a user, who is familiar with the dataset introduced above, and is interested in issuing a set of queries, introduced below. The user poses his data queries using an SQL-like query language syntax in which the elements that are required by the queries but missing in the underlying dataset schema are prefixed with a question mark '?'.

Q_1 : `SELECT title, topic FROM Book.`

Q_2 : `SELECT ?iSBN, title FROM Book WHERE topic = 'Webservices'.`

Q_3 : `SELECT title, author, ?publisher FROM ?Publisher, Book WHERE ?Publisher .?name = Book.?publisher.`

While the execution of Q_1 does not pose any problem, queries Q_2 and Q_3 cannot be entirely evaluated using the introduced database. The reason being that this dataset does not provide all the necessary elements (i.e.,iSBN and publisher do not exist in any of the user tables. Also, the table Publisher is not represented in the schema) for the evaluation of these queries, the missing information can be retrieved from external data sources. With the proliferation of knowledge-sharing communities and the advances in information extraction in turn the construction of databases on the web, we propose in this work to leverage the missing information that could be dynamically obtained from web sources in order to enrich local databases on the fly.

Nowadays, an increasing number of companies and businesses of all sizes are storing some or all of their data on the web, in turn leading to the construction of large data marketplaces on the web as in cloud computing environments, such as Microsoft Azure Data Marketplace [3], Amazon Data sets [1], Oracle Data Cloud [4] and Enigma

[2], as well as in Open Data Initiative. The data stored in these marketplaces is made available and accessible within web services, the so-called data providing (DP) web services or the data services for short. The abundance of available data services has led companies and businesses world-wide to leverage their potential in a number of ways: either implementing their own services to access distant databases possibly located and maintained in different locations or accessing other companies' databases against some fees within the Data as a Service(DaaS) model. In this sense, we explore the possibility of acquiring the missing data from companies' databases through the implemented data services or using DaaS services. Like all the 'as a Service' family, DaaS is built on the concept that the service, the data in this case, is provided on demand to cloud users regardless of their geographic location. This data can benefit a wide spectrum of business models related to biology, chemistry, economics, e-science, etc.

While there exists a large number of data service providers in the market, each one has a different way to describe the services it provides as well as the datasets it supplies regardless of the context in which they will be used. Particularly, data services may define different data structures and semantics for their manipulated data entities than the user/company data structures and semantics. Thus, given a data request, the selection of reliable data services requires a preliminary step of matching the user data request and the service structure. Thereafter, Data queries have to follow the binding patterns of service operations, by providing values for the required input parameters before the service operation can be called. This may be challenging in the context of a service composition.

Actually, individual services may provide only some data and not all the missing information, therefore answering a data query may require investigating different combinations of different data services, which we also call service compositions. Several approaches have been proposed for composing data services [39][72][17][12][66]. The common issue faced while executing the resulting compositions concerns the possible heterogeneity (syntactic, structural or semantic) in the exchanged data between the services involved in a composition. This is not the only issue to be dealt with. As a matter of fact, some service compositions cannot be executed because of the restriction of service patterns. But as some service operations in a service composition may output the mandatory inputs for the execution of other services in the composition, this raises the question as to how to find an executable orchestration between the different component services.

Moreover, the use of a data service is bound to various concerns. Some of them

are technical, specific to the data while others concern the service properties such as the quality of service (QoS) and the cost. The data quality also influences decision making. All these concerns are critical for data service selection. QoS criteria have been extensively studied in favor of data quality. We further argue that the quality of data should be valued as much as QoS to improve the selection decisions. Over the years, the number of data services providing access to similar datasets with varying QoS has significantly increased. As such, most existing works assume that the data provided by different services is the same and complete, thus, the problem becomes how to select appropriate data services so that the quality of the resulting composite service is maximized. Nonetheless, this assumption is unrealistic as long as web services provide access to incomplete data sources. Consequently, the selection of the optimal composition may not completely satisfy the user data queries.

Nowadays, companies are moving away from traditional data-warehouse solutions [11] whereby expensive and time-consuming ETL (Extract, Transform, Load) processes are used, towards data lakes [68]. A data lake is a storage repository that holds a vast amount of raw data in its native format instead of forcing data into a static schema and running an ETL process to fit it into a structured database. As a result, the data is available at a more granular level without losing its details, and schemas are created at a later point when needed. Such data is usually accessed directly through the API of big data platforms such as Hadoop and Spark, or through wrappers. In our PhD dissertation, we go further in this direction and show how data lakes, or more specifically the service lakes, since we are focusing on data services, can be leveraged to answer data queries, taking into account the quality of the services and respecting the budget set by the user. In particular, we focus on the selection and the composition of the relevant data services to fill the possible gaps of information found in companies' data sources and answer user data queries. Moreover, we make sure that we satisfy user requirements in terms of execution cost and service quality. This involves in a first step determining the missing information that is of interest to the user given a data query, then identifying the relevant data services that can be used to populate the missing data, and finally the generation and the execution of the pertinent query plans that produce the best-quality data in response to user queries. In doing so, we automatically model data services as views over the local database schema to lay down the basic foundations for their semantic description, selection and composition and deal with their possible heterogeneity. To the best of our knowledge, our thesis is among the first works to address the automatic specification of service views in the Web services research community.

1.2 Research Questions

The aforementioned issues raise the need for a complete solution to leverage the missing information in user datasets in order to be able to answer the queries he is interested in. This dissertation is guided by the three following main research questions/challenges that need to be addressed in our work:

1) How can we enrich the local data source schema with new concepts that are required to issue user's data queries?

2) A critical step necessary for identifying the authoritative data services that can provide the reliable information, is to understand the capabilities of each available service. Usually, data consumers need to read HTML documentation to collect more information about the service they will choose. This is a time-consuming and tedious task.

3) The new concepts that are used to enrich the local schema which we term 'missing concepts' need to be populated with data instances. These data instances are retrieved by invoking data services. However, it raises the question as to how the schema mappings specifying the correspondences between the new concepts in the local schema and the data services can be defined.

4) The last challenge tackles the problem of processing user queries. Indeed, users may have different needs as to the quality and the cost (both financial and in terms of time) that they are willing to pay for their queries. Here we recall that some of the data services that are provided by the service lake are hosted by cloud providers. Such providers do not supply their services for free, and provide data services with varying data qualities. This raises the question of how user queries can be processed taking into account his requirements in terms of quality and cost.

1.3 Contributions

This thesis provides a novel approach to leveraging the missing information that could be dynamically obtained from data services in order to enrich user data sources on the fly whenever the stored knowledge does not suffice to answer the user queries.

The key contribution of this thesis is a User-centric Data Integration System, that we call *EuDaSL* (***E***nriching ***U***ser ***Da***ta sources in ***S***ervice ***L***akes), to automatically

enrich local data sources. We mention in what follows the three main contributions of this thesis:

- **An approach to automatically generate views for the data services available in the Service Lake:** The determinative factor in selecting and composing data services, particularly DaaSs, is the semantic relationship between their sets of input and output parameters. Thus, the automation of the service selection and composition processes requires the specification of the semantic relationships of data services in a declarative way. This requirement can be achieved through reformulating data services descriptions as views over a global schema, the schema of the company database in our case, following the mediator-based approach. Service views are expressed as conjunctive queries over the relations in the user data source.
- **Algorithms to automatically and dynamically enrich the schema of local databases:** As mentioned above, companies' databases may not represent all the information that cater for the processing of user queries. Thus, missing concepts need to be defined in the schema before proceeding to the extraction of the missing data from external data sources. In this part, we show how the schema of a local dataset can be enriched with new concepts given a workload specifying the user queries. Two algorithms have been developed for this purpose: the first to determine the missing information that is required for the processing of user data queries but not existing in his local data source, the second to enrich the corresponding schema with the missing concepts and relations.
- **A quality-driven service selection and composition approach:** to identify and compose the relevant data services that can be used to leverage the missing information, and to select the best-quality query plans for the enrichment of users data sources. Specifically, we made the following contributions:
  - An algorithm for reformulating queries expressed over the schema of the user data source by replacing the missing concepts in the user data source with invocations to data services. This algorithm adapts existing Local-As-View query reformulation algorithms [50, 49].
  - A knapsack-based algorithm for the selection of the query plans (and therefore the pertinent data services), taking into account the quality of the services, the time and monetary cost limits set by the user.

The main objective of this thesis project being the automatic enrichment of local data sources with information extracted from web sources using data services, the definition

of these services' views over the data source elements remains a crucial step to homogenize the services descriptions, in order to deal with their possible heterogeneity. In a first try, we manually defined views of all the data services in the Service Lake. This manual way is time-consuming and error-prone given the big number of data services that we considered. In a second try, we proposed an approach to automatically generate service views expressed as conjunctive queries over the relations in the user data source. Service views can also be reformulated over a domain ontology introduced as the mediated schema. All these efforts are needed to deal with the semantic and the structural heterogeneities between data services. But what if service providers were willing to make the effort to propose a common ontology for service seekers?

Under this assumption, we present our last contribution MoDaaS, a model-driven framework for the modeling and the description of DaaS services. This framework may encourage providers to adopt a standard model for the description of their data services capabilities and concerns according to a shared ontology, thus enabling them to automatically generate service views in order to assist the integration and data exchange between heterogeneous services. Moreover, the same service description can be reformulated in different formats; RDF descriptions, conjunctive queries, OWL-S descriptions, etc.

1.4 Outline of the Thesis

The remainder of this thesis is structured as follows:

- **Chapter 2** introduces the basic notions and the different concepts needed for the understanding of our work. We also present a state-of-the-art on data integration and service composition approaches existing in the litterature. These approaches are discussed and details of some example works are explained to address where our work stands in the literature.
- In **Chapter 3**, we first provide an overview of the different contributions in this dissertation and the overall architecture of our user-centric data integration approach. Then, we describe the enrichment of the schema of user data sources. Two algorithms have been proposed to determine the missing data, given a set of data queries, and to enrich the schema of the user data source. We demonstrate the proposed algorithms with one scenario from the motivating examples in Section 1.1.

- In **Chapter 4**, we investigate the Service Lake and identify the relevant data services that can be used to leverage the missing data. Afterwards, we present the different aspects of evaluation of our proposal; providing information about the prototype implementation, then an evaluation of the proposed composition solution is presented by comparing it with existing composition frameworks.
- **Chapter 5** addresses the specification of service views over the relations of the local data source in the Service Lake. To the best of our knowledge, this is the first work to deal with the automation of service views definition.
- Last but not least, **Chapter 6** describes our model-driven framework for the modeling and the description of Data-as-a-Service to assist service selection and composition.
- Finally, we sum up our contributions in **Chapter 7**. We conclude this last chapter with our ongoing works and some perspectives that we aim to achieve in short, medium and long terms.

Chapter 2

Background & State of the Art

Contents

Our work has taken shape in the context of a rich and interesting literature focused on data integration and service composition. This chapter is devoted to give some preliminary notions about these two research areas. In particular, we first explain the

principles of data integration and we introduce the existing integration approaches. We then give some basic definitions and notions related to data services and service composition, that will be used throughout the manuscript. In a second part of each section, we analyze and discuss the main related works.

2.1 Data Integration

Data integration is the process of combining data residing at different, autonomous and possibly heterogeneous data sources and providing the user with a unified view of these data [47]. The main aim behind is to allow users coming with a single data query to find relevant data no matter which database provides it.

A first issue, from a technical viewpoint, the difficulty comes from the lack of interoperability between the data sources, that may use a variety of formats, specific query processing capabilities or different protocols. However, the real bottleneck for data integration comes from the semantic heterogeneity between the data sources, such as companies organize their data using different schemas.

Definition 1 *Semantic Heterogeneity refers to the differences or similarities in meaning and interpretation of data values, when the schema of different databases for the same domain are developed by independent parties [41]. In other words, semantic conflicts among information systems occur whenever information systems do not use the same interpretation of the information.*

Accordingly, two schema elements in two local data sources can have the same intended meaning, but different names. For instance, in example 1 in Figure 2.1 , different names were attached to the same concept (SSN versus ID) in the two schemas of Student and Grad-Student, respectively. Thus, during integration, it should be realized that these two elements actually refer to the same concept. Alternatively, two schema elements in two data sources might be named identically, while their intended meanings are incompatible. This is the case in example 2 (cf. Figure 2.1); title in the table Book refers to a title of a book, whereas the attribute title in table Album refers to the title of an album. Hence, these elements should be treated as different during the integration.

Different integration approaches have been proposed in the litterature, some are widely used and some are less so, having failed to achieve the basic requirements of

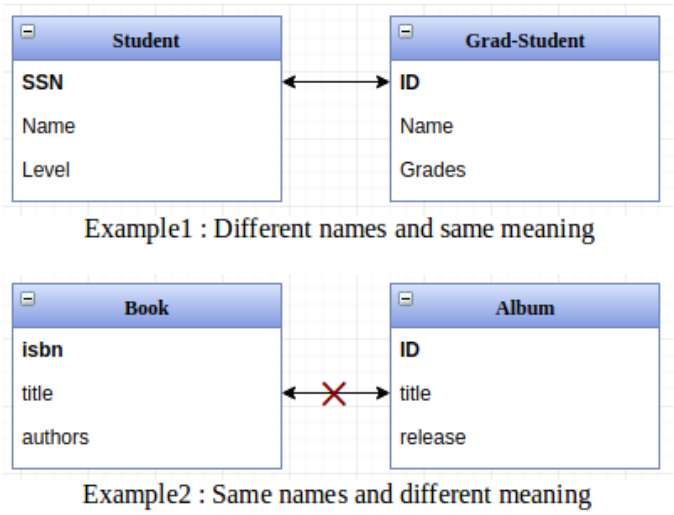


Figure 2.1 – Possible Semantic Conflicts

data integration. The two most important architectures for building a data integration system are: (1) The *Materialized integration*, where data is physically copied (with or without transformation) to a dedicated storage, e.g., Data Warehousing, Data Lakes. and (2) The *Virtual integration*, in which data remains in its original sources and is accessed during the query execution. In the following sections, we discuss the integration architectures, its strengths and weaknesses. We also discuss in the section thereafter some integration systems proposed in the literature.

2.1.1 Materialized Integration

The Materialized Approach derives its basis from traditional data warehousing techniques. Data from heterogeneous distributed information sources is gathered, mapped to a common structure and stored in a centralized location. Warehousing emphasizes data translation, as opposed to query translation in mediator-based systems. Figure 2.2 shows an overview of the materialized architecture. In fact, warehousing requires that all the data loaded from the sources be converted through data mapping to a standard unique format before it is stored locally. In order to ensure that the information in the warehouse reflects the current contents of the individual sources, it is necessary to periodically update the warehouse. In general, the execution of data queries posed to a data warehouse is not expensive and takes very little time to resolve. That is because the data warehouse has already done the major work of extracting, converting and combining data and Database managers put a lot of thought to make it effective and efficient.

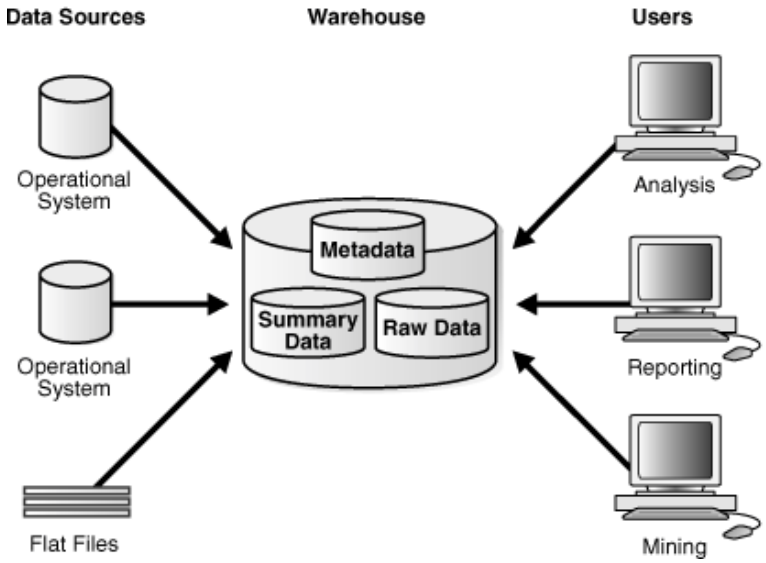


Figure 2.2 – Materialized Integration Architecture

The problem being with this architecture is that the information stored in data warehouses need always to be updated. That is because of the way data warehouses work such as they pull information from other databases periodically. If the data in those databases changes between extractions, queries to the data warehouse will not result in the most current and accurate views. Consequently, If the data in a system rarely changes, this architecture is not the most appropriate for this sort of applications.

2.1.2 Virtual Integration

In the virtual model, the integrated database consists purely of the virtual view definitions, as querying interface and a querying engine. No data is stored in the integrated database. Users ask queries over the virtual views, and a query engine translates these into queries over the remote data sources. Each data source, possibly through its wrapper, answers the query, and sends results back to the integrated database. The query engine is responsible for combining these results into an answer to the original query.

To allow the mediator to decide which data to retrieve from each source and how to combine them into the unified view, the administrator of the integration system has to specify the correspondence between the local schema of each source and the global schema through mappings.

Definition 2 ***Global Schema****, also called mediated schema, serves as a unique entry*

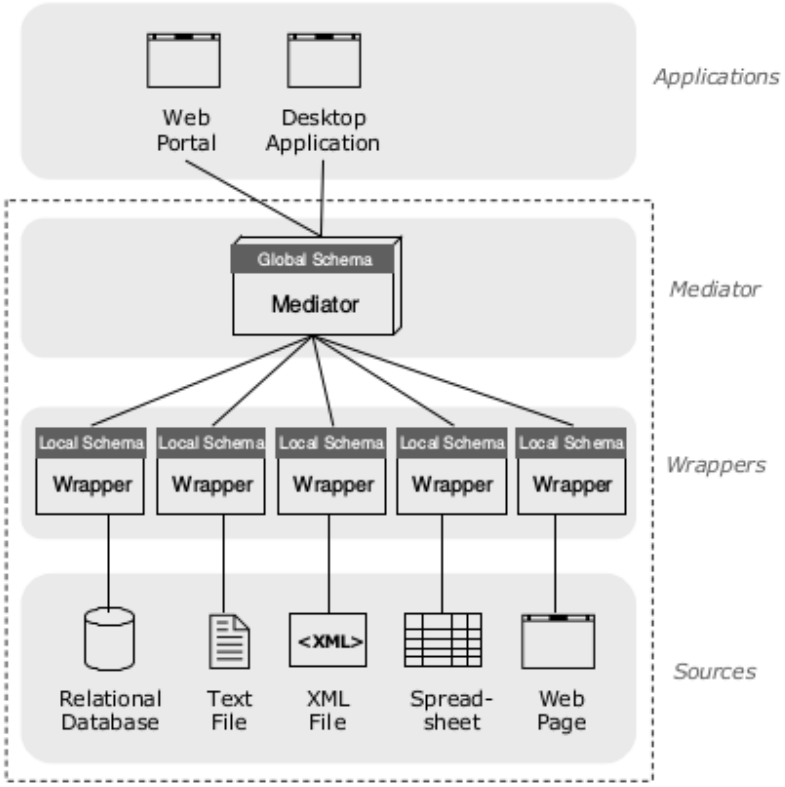


Figure 2.3 – Virtual Integration Architecture

point on which global queries are posed by users, providing a reconciled, integrated, and virtual view of the underlying sources.

A main issue is then to specify the relationships, namely semantic mappings, between the schemas of the data sources and the global schema. Based on these mappings, one can answer queries over the global schema using queries over the data sources. Typically, query answering in the mediator approach is performed as follows. First, independently of the data in the sources, the user query posed over the global schema is transformed into local queries that refer to the schemas of the data sources. A global query combines the data provided by sources. Queries are optimized and transformed into query plans. The local query plans are executed and their results combined by the global query plan.

Relationship between the global schema and the local data sources (and their local schemas) is specified at the mediator level.

Two basic approaches for specifying the mappings in a data integration system have been proposed in the literature, called local-as-view (LAV), and global-as-view (GAV),

respectively [42]. Each approach focuses on a particular part of the overall system and has its advantages and disadvantages.

GAV, Global as View Approach In a global-as-view approach, the focus is on the global schema. The global schema is described in terms of the local schemas, such as the content of each element of the global schema should be characterized in terms of a view over the data sources, and specifies how to obtain tuples [1] of the global relation from tuples in the local sources. The main advantage of GAV is its conceptual and algorithmic simplicity. Also, as long as the data sources remain consistent, the global-as-view approach works well. However, adding or removing data sources to the system is problematic, because it affects data across the system as a whole.

LAV, Local as View Approach The local-as-view technique takes the opposite approach, focusing on the data sources. The content of each source s should be characterized in terms of a view over the global schema. The goal is to define the global schema in such a way that individual definitions do not change when data sources join or leave the integration system except for the definitions of the sources that are involved in the change. LAV mappings enable quite fine-grained descriptions of the contents of data sources. As long as the global schema remains constant, it is easy to add or remove data sources to the system. Nevertheless, changing the parameters of the global schema is difficult. If we want to analyze the data sources in a new way, we have to redefine the entire system.

To conclude, in the GAV approach, every entity in the global schema is associated with a view over the source local schema. Therefore querying strategies are simple, but the evolution of the local source schemas is not easily supported. On the contrary, the LAV approach permits changes to source schemas without affecting the global schema, since the local schemas are defined as views over the global schema, but query processing can be complex. To overcome the limitations of both integration strategies, [38] proposed the Global and Local As View (GLAV) approach, which combines the expressive powers of both GAV and LAV approaches, allowing flexible schema definitions independent of the particular details of the sources.

For designing the mappings, the distinction made in the mediator model between local and global relations does not always make sense, such each peer relation may play the role of both a local relation and a global relation at different times. Therefore,

[1] In computer science, a tuple (also called record) is one of the simplest data structures, consisting of two or more values or variables stored in consecutive memory positions.

the notions of GAV and LAV mappings are relaxed to the more appropriate symmetric notion of GLAV mappings.

GLAV, Global and Local as View Appoach GLAV languages can trivially express both GAV mappings and LAV mappings by assigning to a query expressed over the global schema a query returning a single global relation or to a query posed over the local schema a query asking for a single local relation, respectively. In other terms, GLAV uses flexible first-order sentences such that it allows a view over source relations to be a view over global relations in source descriptions. Thus, GLAV can derive data using views over source relations, which is beyond the expressive ability of LAV, and it allows conjunctions of global relations, which is beyond the expressive ability of GAV.

2.1.3 Related Work

Over the past decades, considerable academic and commercial efforts have been made to deal with data integration, most of which are being surveyed in [43], [15] and [84] such as KARMA [79], TSIMMIS [27], MOMIS [19], SIMS [13], the Information Manifold [48], HERMES [6] and ANGIE [65], etc. In this thesis, we restrict our attention to describe view-based data integration systems.

1) **TSIMMIS** [27]

   TSIMMIS, The Stanford-IBM Manager of Multiple Information Sources, is a project whose main goal is to develop tools that facilitate the rapid integration of heterogeneous information sources including both structured and unstructured data. Figure 2.4 presents a high-level overview of the overall TSIMMIS architecture with all its components. TSIMMIS is a typical example of the mediation approach in which wrappers stand above each data source converting application queries into source specific queries and translating obtained data into a common data model. Mediators are used for the integration part refining in some way information from one or more sources.

   To properly describe the common data model, TSIMMIS introduced its own simple and self-describing object model called OEM (Object Exchange Model) and an SQL-like query language for requesting OEM objects named Lorel Query Language. Both mediators and wrappers export the same interface taking a query as an input and returning OEM objects. Such a unified approach allows to access the data sources transparently either directly from the wrappers or the

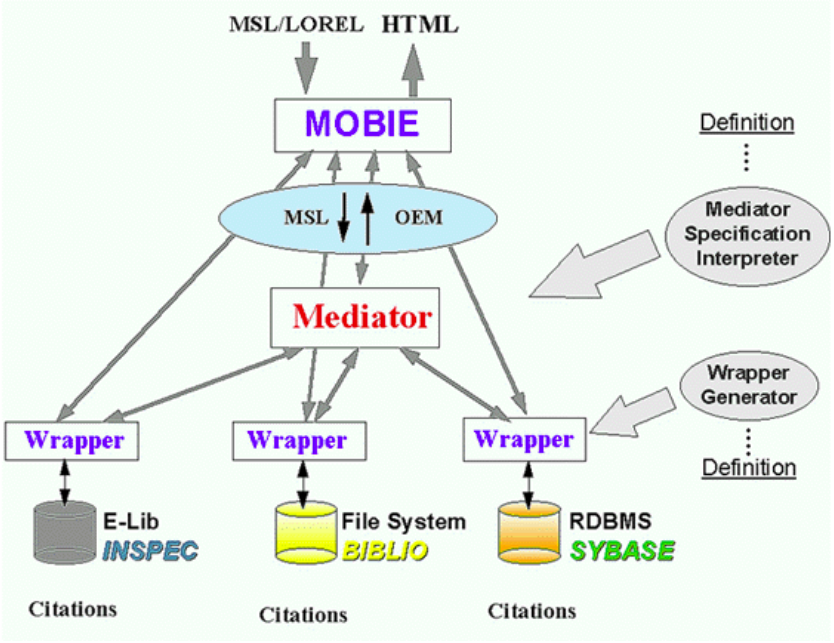


Figure 2.4 – High-Level Overview of the Overall TSIMMIS Architecture

mediators. Hence, a new data source becomes useful as soon as a wrapper is supplied. TSIMMIS concentrates especially on flexibility, so it is well prepared for unexpected occurrences of heterogeneity. Accessing very diverse and different information which may frequently change its content or meaning is the basis of this system. The downside of such an approach is that in some cases, the integration must be performed manually by the end user. So, as the authors of this project say, TSIMMIS does not perform fully automated integration but rather provides a framework and tools to assist the users in information processing and integration efforts.

2) **MOMIS** [19]

MOMIS (Mediator EnvirOnment for Multiple Information Sources) is an Open Source Data Integration system able to aggregate data coming from heterogeneous data sources (structured and semi-structured) in a semi-automatic way. MOMIS builds a Global Schema, of several (heterogeneous) data sources, and allows users to formulate queries on it. It follows a Global-As-View (GAV) approach for the definition of mappings between the GS and local schemas: the GS is expressed in terms of the local schemas. MOMIS performs data integration following a virtual approach that preserves the autonomy and security of the original data sources. This gives rise to a Global Virtual View (GVV), in the form of global classes and global attributes, which represents a unified and

integrated view of data residing in the different local data sources. Figure 2.5 illustrates the generation process of the global schema:

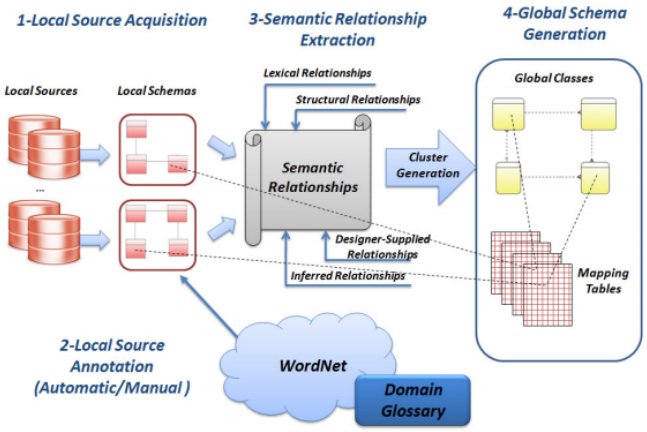


Figure 2.5 – MOMIS Data Integration Process

- **Local Source Upload:** the integrator designer exploits the wrapper tool to logically extract the schema of each local source and convert it into the common language ODL.
- **Local Source Annotation:** the designer is asked to annotate the local sources, i.e. to associate to each class and attribute names one or more meanings w.r.t. the lexical database WordNet.
- **Semantic Relationships Extraction:** starting from the annotated local schemas, MOMIS derives a set of intra and inter-schema semantic relationships in the form of: synonyms (SYN), broader terms/narrower terms (BT/NT) and related terms (RT) relationships. The set of semantic relationships is incrementally built by adding: structural relationships (deriving from the structure of each schema), lexical relationships (deriving from the element annotations, by exploiting the WordNet semantic network), designer-supplied relationships (representing specific domain knowledge) and inferred relationships (deriving from Description Logics equivalence and subsumption computation). This is performed by the Global Schema Designer tool.
- **Global Schema generation:** starting from the discovered semantic relationships and the local sources schemas, MOMIS generates a GS consisting of a set of global classes, plus Mapping Tables which contain the mappings to connect the global attributes of each global class with the local sources attributes . The GS generation is a process where classes describing the

same or semantically related concepts in different sources are identified and clusterized into the same global class.

The designer may interactively refine and complete the proposed integration result through the GUI provided by the Global Schema Designer tool.

3) **KARMA** [79]

KARMA is a data integration system that enables users to quickly and easily integrate data from a variety of data sources including databases, spreadsheets, delimited text files, XML, JSON, KML and Web APIs. Users integrate information by modeling it according to an ontology of their choice using a graphical user interface. Karma learns to recognize the mapping of data to ontology classes and then uses the ontology to propose a model that ties together these classes [80]. The generated model can then be adjusted by the users, who can also transform the data in different formats as needed in order to normalize data expressed in different formats and to restructure it during the integration process. Once the model is complete, users can publish the integrated data as RDF or store it in a database. The overall architecture of KARMA is shown in Figure 2.6.

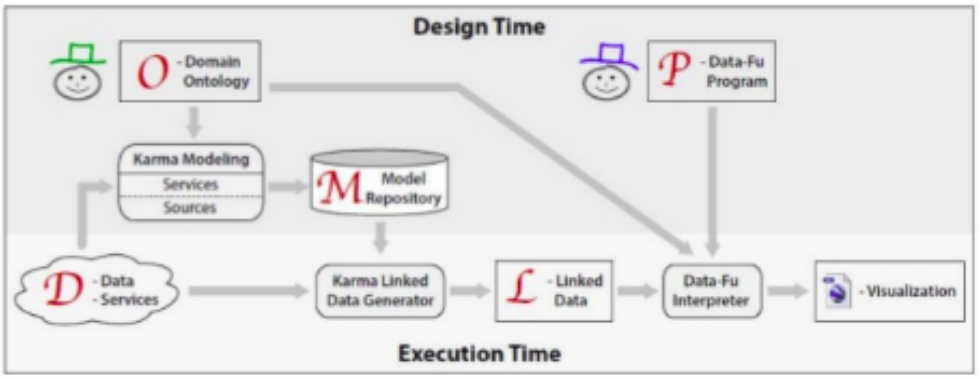


Figure 2.6 – KARMA Architecture

4) **IM, Information Manifold** [48]

Information Manifold is a system for browsing and querying of multiple network information sources. The system demonstrates the viability of knowledge representation technology for retrieval and organization of information from disparate information sources. The language for representing contents of information sources is a combination of Horn rules and concepts from the CLASSIC description logic. The descriptive terminology in CLASSIC contains unary relations

called concepts, which represent classes of objects in the domain, and binary relations called roles that describe relationships between objects concepts. These roles can be either primitive or complex. Integrity constraints are used to specify types of the attributes of the domain relations. The knowledge base contains ontologies for representing various aspects of the domain. In particular, it represents physical properties of information sources such as the URL addresses, the protocols used to access them and their internal structure. The knowledge base also consists of a rich topic hierarchy, ontologies for representing properties of people, organization, geographic locations and time. The queries answering process proceeds as defined by the LA V approach. The expressive power of horn rules is necessary in order to model information sources that are relational databases (Negation and statements describing relationships between the information sources). It is a rules-based system. It verifies data consistency but remains unable to infer and analyse new facts. In addition, it can use constraints by manipulating relational data sources only.

5) **SIMS** [13]

SIMS (Services and Information Management for decision Systems) is a data integration system of information from various information sources. The information sources handled include both databases and knowledge bases, and other information sources (e.g., programs) could potentially be incorporated into the system. SIMS accepts queries in the form of a description of a class of objects about which information is desired. This description is composed of statements in the LOOM knowledge representation language [40], [54]. The system requires a model of the application domain and a model of the contents of each of the information sources. Then, given a query, the system generates and executes a plan for accessing the appropriate information sources. Before executing a query, the system first reformulates the individual subqueries to minimize the cost and the amount of intermediate data that is processed. Then the subqueries are executed, exploiting any parallelism in the plan. SIMS currently integrates information from data stored in nine Oracle databases and information stored in a LOOM knowledge base. The system uses the LOOM Interface Manager (LIM) [62] to retrieve data from the Oracle databases and then processes all the data in LOOM. The plan for selecting and accessing the various information sources is generated using the Prodigy planning system [57]. The resulting plan is reformulated using a set of special purpose algorithms for semantic query optimization over multiple database queries. The completeness and complexity of rewriting algorithms were not approached.

6) **HERMES** [6]

HERMES (A heterogeneous reasoning and mediator system) is a system for semantically integrating different and possibly heterogeneous information sources and reasoning systems. based on two main aspects in creating the mediation system. The first, the Integration domain: it consists to manage the addition of new data source or a reasoning system to an existing mediation system so that the new data, data representation or the corpus of new reasoning algorithms are accessible by various mediators. The second, the Semantic Integration. It is the process of determining methods to resolve conflicts and retrieve data and thus compose the result information from the individual data sources. HERMES is a federated database system. It is not based on the semantic description of a scope or content of different databases. Its goal is simply to combine queries posed to various database management systems. It integrates the domain of relational databases, the domain of spatial databases, the domain of text databases and the domain of pictorial databases. Each domain is considered as database abstraction and contains software packages. It consists of three sets: the values, functions on values and relationships on the values of data objects. The mediator language is based on rules. Its syntax is similar to Prolog. The access to a domain incorporated by HERMES is done via a special set of predicates.

7) **IBIS** [25]

IBIS (Internet-Based Information System), semantic data integration approach that fully exploits all available information, including integrity constraints for query answering. It uses a relational global schema to query the data at the sources following the GAV approach. IBIS is able to cope with a variety of heterogeneous data sources, including data sources on the Web, relational databases,and legacy sources. Each non-relational source is wrapped to provide a relational view on it. Also, each source is considered incomplete, in the sense that its data contribute to the data integration system. The key issue of IBIS is that it allows the specification of integrity constraints in the global schema ans since data sources are autonomous and incomplete, the extracted data in general do not satisfy the constraints. Therefore, IBIS adapts and integrates the data extracted from the sources making use of the constraints in the global schema, so as to answer queries at best with the information available. In this way, the constraints over the global schema allows to obtain additional answers that would not be provided by the standard unfolding strategy associated with GAV data integration systems. Furthermore, IBIS exploits and implements techniques developed for

querying sources with binding patterns in order to retrieve the maximum set of answers, taking into account intentional knowledge holding on the sources (in particular, integrity constraints) to limit the number of source access.

8) **ANGIE** [66]

ANGIE is a data integration tool that allows users to automatically enrich local RDF knowledge bases with data provided by SOAP or RESTful services. When a user poses a query, if this latter can not be answered by the local database alone, ANGIE identifies and calls the appropriate data services to retrieve the missing information and integrate it into the local database.

The authors in [65] assume that service definitions are given and are incorporated in the knowledge base as function definitions. These latter are modeled as RDF graphs, as well as data queries, where edges are partitioned into input and output edges. Figure 2.7 shows an example of a function definition ,on the right side, which given a writer name, it returns its books. The input edges have to be

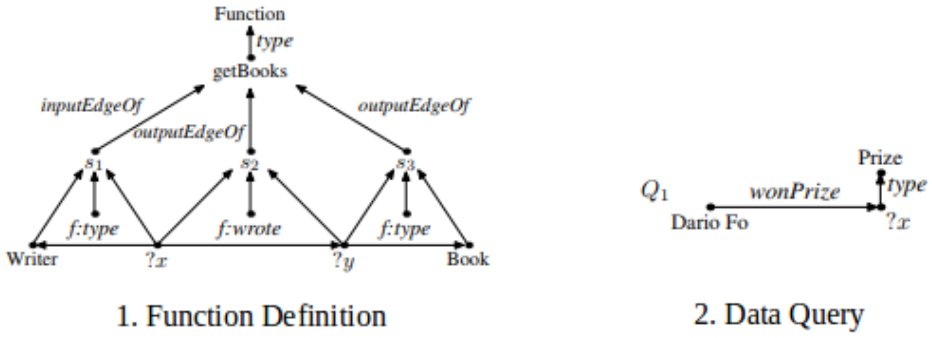


Figure 2.7 – Data Model Adopted in ANGIE

fulfilled before the function definition can be called (i.e., input parameters) while the output edges present the function call results (i.e., output parameters). This way, the function definitions are integrated completely into the knowledge base and become first class citizens of the knowledge base. A function call is then a partial instantiation of the function definition that binds all input parameters (i.e., instantiate some variables of the function with entity or relation names. Other variables of the function definition are simply given new variables names).

N. Preda et al. proposed an algorithm to automatically generate appropriate service calls. Given a SPARQL query, the algorithm translates the query into a sequence of function compositions, including artificial functions extracting data from the knowledge base. This enables the system to always make use of the

local knowledge first. The algorithm relies on a depth-first-search strategy to expand the query cover and computes and outputs the rewritings for a query in the spirit of the chronological backtracking strategy used in Prolog. Note that a data query can be answered by several function compositions, the proposed algorithm explores all of the possible rewritings in order to retrieve a larger set of results, as some results are returned only by particular function compositions. Then, it orders the service calls according to a principled cost model (i.e., the cost of functions instantiation) and prioritize those functions that are expected to yield results faster or with lower execution costs. Once the web calls are executed, their results are transformed into RDF and are dynamically added to the local knowledge base.

2.1.4 Discussion

In a nutshell, several integration systems are built on the notion of mediators, instructing the system exactly how to retrieve elements from the source data sources. This requires constructing a global schema on which global queries are posed by users, however, if any new sources join the system, considerable effort may be necessary to update the mediator. Other efforts as [13] and [50] construct a general domain model (under an information manifold) that encompasses the relevant parts of the data sources scheme where the description of different data sources is done independently from the queries that are subsequently asked on them. Then, the integration problem is shifted from how to build a single integrated schema to map between the domain and the data source descriptions.

While these approaches reduce the user's effort to perform data integration tasks, users queries must be formulated over the mediated schema (either the integrated schema or the domain model), therefore, users are required to pick up complementary data sources to interrogate in order to get sufficient answers to their queries. However, the interaction is not guaranteed to yield a non-empty result set.

Our work differs from past integration systems in that we propose an active data integration approach where queries are posed over the user data source schema s/he is interested in. Furthermore, users can query information that does not exist in their datasets and it is up to our system to enrich the initial schema and leverage the missing information from External data sources.

ANGIE is perhaps the closest work to ours in that it attempts to enrich knowledge databases by leveraging the missing information from web sources. However, this work

assumes the existence of a global schema for both data and services. This assumption makes ANGIE domain specific and not suitable for general purpose queries. In ANGIE, the enrichment of the knowledge base is done only at data instance level, whereas, in our approach, it is also possible to enrich the schema of the data source by defining new concepts and relations additionally to data instances. The enrichment of the schema is applied automatically by the system without user demand or human intervention.

In the following, we draw a comparative table of the data integration approaches described above and highlight the positioning of our work on data integration among them. Table 2.1 provides a global overview of the data model, the methods used and the functionalities provided by each data integration system.

System	Integration Approach	Data Model	Views Definition Methods	Schema Enrichment	Data Enrichment
SIMS [13]	LAV	Rules-based global schema	manually	manually	manually
TSIMMIS [27]	GAV	Rules-based global schema	manually	manually	manually
IBIS [25]	GAV	Classes-based global schema	manually	manually	manually
MOMIS [19]	GAV	Rules-based global schema	manually	manually	manually
HERMES [6]	GAV	Classes-based global schema	manually	manually	manually
IM [48]	LAV	Classes-based global schema	manually	manually	manually
KARMA [79]	GAV	Classes-based global schema	manually	manually	manually
ANGIE [66]	GAV	Graph-based global schema	manually	manually	automatic
EuDaSL	LAV	Graph-based global schema	semi-automatic	automatic	automatic

Table 2.1 – Comparison of Most Known Data Integration Approaches

2.2 Service Composition

In this section, we first introduce the Service-Oriented Architecture (SOA) which is the grounded in the idea of service composition, the data providing web services and the main types and architectures of web services. Then, we describe the service composition process, giving the main steps . Next, a discussion between different automatic service composition approaches is raised.

2.2.1 Service Oriented Computing

Service Oriented Computing (SOC) has emerged and been widely accepted as a distributed computing paradigm over the Internet. SOC provides mechanisms for publication, discovery, selection, and composition of Web services, and thus facilitates the integration of complex and heterogeneous software components. In particular, it enables applications written in different languages and running on different platforms, to communicate among them and be accessed by the same users. In other words, Service Oriented Architecture (SOA) principles enable services to be used by other services or programs.

Modern enterprises are increasingly embracing the service-oriented paradigm to provide interoperable and programmatic interactions with their internal systems. As discussed in [16], Web services can generally be categorized into two types: (i) effect-providing (EP) services, which implement organizations functionalities and (ii) data-providing (DP) services, which allow accessing the data sources of organizations.

2.2.2 Web Services Standards

Generally speaking, a web service is a software artifact, delivered over the Internet, that interacts with its clients in order to perform a specified task. A client can be either a human user, or another service. When executed, a service performs its tasks by directly executing certain actions, possibly interacting with other services to delegate to them the execution of other actions. In order to address the Service Oriented Computing paradigm from an abstract and conceptual point of view, we start by identifying several facets, each one reflecting a particular aspect of a service during its life time.

- The service schema specifies the features of a service, in terms of functional and non-functional requirements. Functional requirements represent what a ser-

vice does. All other characteristics of services, such as those related to quality aspects, privacy and security, performance, transactions, etc. constitute the non-functional requirements.

- A service instance is an occurrence of a service effectively running and interacting with a client. In general, several running instances corresponding to the same service schema may co-exist, each one executing independently from the others.

2.2.3 Data Providing Web Service

Besides using Web services to provide access to corporate applications and software assets over the Web, a recent trend has been to use Web services as a reliable means for data publishing among organizations [26]. Such services are known as Data Providing(DP) Web Services or the Data services for short.

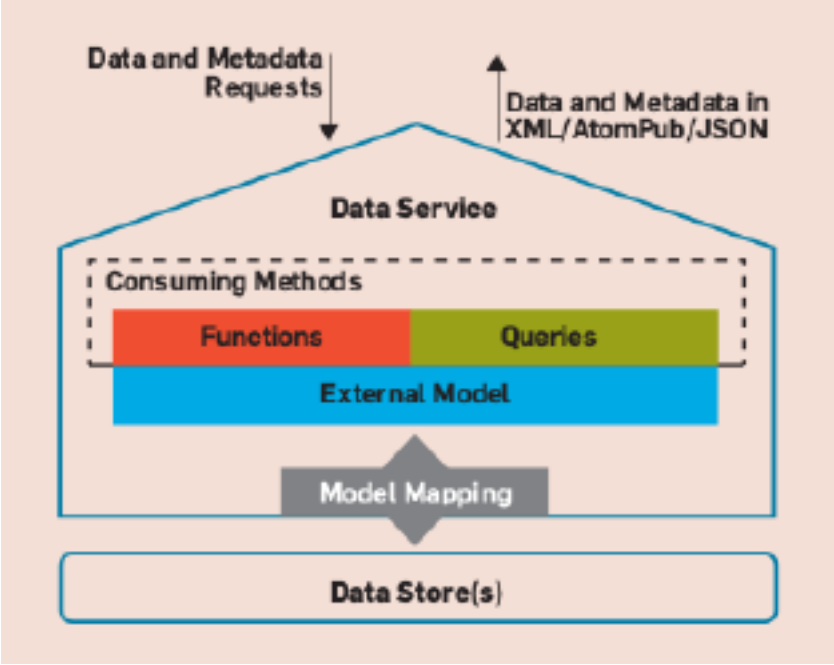


Figure 2.8 – Data Service Architecture

Data services differ from traditional Web services in that they serve as “fronts” for data and are based on a richer model of that data. Nowadays, companies and world-wide enterprises are saving their data on the web, thereafter providing services-based access to their data through data services. The introduction of data services has allowed to shield users from having to directly interact with the various data sources that give

access to business objects (i.e., customers, orders, invoices, etc.) and enabled them to focus on the business logic only. In fact, the invocation of a data service results the execution of a query on the schema of the data source behind, but users cannot directly access the data and observe the data source schema that the service may use internally. These services can be accessed only through an encapsulated API (i.e., when such a service is invoked, he accepts a request as an input with a specified format and returns requested data as output); answers to queries are returned in a semi-structured format (XML or Json).

Data providers expose certain data resources accessible via service operations (e.g., based on WSDL or REST APIs). Using these operations, the data consumer can select certain data resources via service interfaces. In the following, we describe the main web service architectures.

2.2.4 DaaS, the Data as a Service model

While data services were initially conceived to solve problems in the enterprise world, the cloud is now making this type of services accessible to a much broader range of consumers through the Data as a Service model (DaaS).

Data as a Service (DaaS) is an emerging cloud computing service that provides data on demand to consumers across various cloud platforms via different protocols over the Internet. The same benefits that come with any major Cloud-computing platform also apply to the Data-as-a-Service space such as the avoidance of "vendor lock-in", ease of administration and collaboration, global accessibility, automatic updates and the cost effectiveness.

The main exception for DaaS providers is that their benefits reach for and are deep into the world of Data Management. In fact, utilizing DaaS does not only support data access from anywhere at anytime but also reduces the cost of data management. Traditionally, companies housed and managed their own data within a self-contained storage system. With the increasingly large amount of data, data processing and analysis in a large dataset becomes too complex to be effectively processed by traditional approaches and needs high computing capacities. Also, moving the data being stored in local companies data sources is a serious difficulty due to time transfers and network link limitations.

Besides limitations for the data sharing, traditional approaches also do not achieve

to fully separate/decouple software services from data and thus impose limitations in interoperability. The introduction of DaaS has allowed to shield users from having to directly interact with the various data sources that give access to business objects (i.e., customers, orders, invoices, etc.) and enabled them to focus on the business logic only. The main idea is about offloading the risks and burdens of Data Management to a third-party Cloud-based provider, such as the bulk of data access is primarily controlled through the data service itself. This adds a robust layer of security and improves data quality.

2.2.5 Web Service Architectures

Web services are of two kinds: Simple Object Access Protocol (SOAP) and Representational State Transfer (REST).

SOAP is used to wrap XML messages exchanged between service consumer and provider. It is based on sending an XML message to a service, in a specific format, and receiving an XML response in another specific format. The message can be sent across different transports, including HTTP, FTP, SMTP, etc. The specification does not dictate the transport over which the message should be sent, but most implementations send the XML message over HTTP. Because of how extensible it is, the SOAP message is rather large: it wraps its message in what is called the SOAP Envelope and the Envelope has a SOAP Header and a SOAP Body.

REST , on the other hand defines resources and then provides access to those resources through the HTTP verbs (GET, POST, PUT, DELETE, HEAD, and so forth.). In RESTful APIs, resources are identified by URIs.

WSDL is an XML-based language used to provide structured descriptions for services, operations and endpoints. It mainly provides a machine-readable description of how the service can be called, which parameters it expects, and which data structures it returns.

2.2.6 Service Composition

In general, service composition is the method to create new services or applications by composing existing services. The output of service composition process is a composite service which delivers desired functions. Since users requirements are complex, an execution of only one service is unlikely to fulfill the users needs. Thus service composition plays a part in the procedure of assembling the existing services if relevant to the users needs.

2.2.6.1 Web Service Composition Life Cycle

In this section, we discuss how a composite service can be built. Generally, service composition process occurs in three subsequent phases, described in the following:

- **Composition:** This phase deals with synthesizing the composition schema. Given a complex requirement, the composition schema designer decomposes the requirement to build up the composition schema or workflow schema. The schema consists of component services and control and data flow specification. The control flow specification sets up the order in which the component services should be invoked.
- **Selection:** This step finds and matches the advertised service specifications and the component service functions. Relevant services can be discovered from a service registry after the composite schema is formed.
- **Orchestration & Execution:** service execution governs the order in which services are invoked, and the conditions under which a certain service may or may not be invoked. In this phase, the executable composite service is deployed to create its instance. Next, the composite service instance then allows an invocation by end user then is executed by process execution engine.

Manual discovery and composition of relevant data services is time-consuming such as data consumers has to visit DaaS providers one by one to look for the services they provide and their datasets' descriptions. Furthermore, given that the number of DaaS providers as well as the data services could be very large, this way of selection is tedious. Different challenges exist and several approaches have been proposed for the composition of data services. The section that follows examines the main challenges that can appear when selecting, creating and executing a data service composition

answering the user query. Then, we review some of the most important proposed composition approaches.

2.2.7 Challenges and Issues of Data Service Composition

Automatic data service composition is one of the critical research challenges of service-oriented computing. Still the research is going on finding the appropriate services answering users' queries from a set of candidate services, building of composite services when no individual service satisfies the users' requirements, and the execution of service compositions. However, no approach so far has focused on all the potential challenges that need to be addressed beforehand. The most important issues will be discussed in the following paragraphs.

- **Dealing with heterogeneous data formats, description and interaction models:** Data services and DaaS in particular have become a standard way for data publishing and sharing on the web. Nevertheless, there exist no well-defined ways to describe their properties (functional and non-functional properties) and their associated data assets. Indeed, data services are typically published in the most convenient way for the publisher; each data provider has his own way to describe the services and the data he provides, mostly using HTML documentation. Often, users have to go manually through an overwhelming number of data services to read their documentations and reformulate their descriptions according to a chosen data model with the aim to make service descriptions machine readable and thus enable the automatic selection and composition. Therefore, enormous number of service description languages such as WSDL, OWL-S and SAWSDL have been proposed for the (semantic) description of services' capabilities to describe the functional and non-functional properties of data services with a machine-readable way in order to enable the automatic discovery and thereafter to provide better results compared to the traditional discovery method. However, this large variety of languages has resulted an overlap and differences in their capabilities at conceptual and structural levels. Such differences affect discovery, selection and composition techniques. Thereafter, the lack of well-defined and standard machine-readable model hinders the automatic discovery and composition of data services. Thus, mechanisms to facilitate data processing and giving data an homogeneous structure are needed for data consumers to make the most of data services. This also raises a number of issues during the selection and the composition of the relevant services answering users' data queries. In a nutshell,

the automatic composition of data services implies to deal with heterogeneous terminologies, data formats, and interaction models.

- **Modeling data concerns:** More and more web services with similar function attributes but different QoS are available. Thus, service composition requires a high degree of interoperability among the component data services such as it should consider the functionality of the participating services (I/O parameters), the data that is passed between these services (data flows), the quality of the data it provide, the quality of the composite service as a whole (QoS) and the execution pattern of these services (API templates). Current (semantic) Web service languages describe the syntactical aspects of a Web service, and therefore, the result is rigid services that cannot respond to unexpected user requirements and changes automatically, as it requires human intervention. The approach must be able to choose the best available service that fulfills the composition requirements. Some of the proposed approaches proposed an automatic selection based on QoS, rating, and user feedback. However, QoS and rating do not ensure that the best services will be selected. There is a need for richer description handling also the description of the data quality additionally to the service properties. To conclude, even semantic languages initiatives do not enable automatic service composition techniques and none of them achieve fully automated service discovery, selection, or composition.

All the aforementioned challenges contribute to the same high-level goal to (a) provide rich and flexible model for the description of all service properties (functional and non-functional properties) and (b) automate the discovery, the selection, and the composition processes of data services, those that fulfill the user's requirements (service and data qualities). We devote the following section to review the related works achieved in the literature.

2.2.8 Related Work

As mentioned above,publishing, discovery, and selection mechanisms as well as heterogeneity and limitations of semantic languages have a major impact on the effectiveness of service composition approaches. In the following, we start by describing the efforts made to deal with the first of these challenges; the publishing and the description of data services.

2.2.8.1 Modeling and Description of Data Services

Describing data services is still a challenging issue, since service description languages such as WSDL, OWL-S and SAWSDL are not rich enough to describe the underlying data model of a service. In the literature, little effort have been spent on supporting the description and the modeling of data services.

In [82] a proposal, called DEMODS, to describe a general model to cover all basic information of DaaSs is presented. In DEMODS, the data service API is decoupled from the data asset description. It maily relies on external information models to describe the data assets, but it does not provide any special support for them, beyond links to the description documents.

Mashroom [44] uses nested tables to model data services and provides a family of tools to transform HTML, XML, and JSON to the data model. In fact, various data sources are encapsulated as data services with nested tables as their unified data model both for internal processing and for external uses. Users can operate on the nested tables interactively, however it is not the case for data applications.

Other frameworks have been developed for publishing information about Web services' capabilities but they neglect the publishing of data concerns. A generic framework for the evaluation and publishing of data concerns, associated with data exposed through DaaS, was proposed in [78]. The authors have analyzed most of DaaS concerns in [77] in details. We go further in this direction and we extend their model and implement it using the MDE capabilities. Furthermore, we implement a generative tool to generate the corresponding service views over a domain ontology.

Many works have proposed solution for automatic web service composition approaches such they regard DaaS as RDF views. Our work complements these efforts by providing an integrated framework to automatically define services' views given their descriptions. The Open Data as a Service (ODaaS)[74] approach uses multi-level modeling to construct open data applications . It consists of a set of domain models and meta-models and a library of 'injectors' to import data from heterogeneous source. The domain descriptions are the classification of concepts and their successive refinements by means of the multi-level modeling, that allows mapping data in some format into a semantically rich model. The data modeling is organized based on generic domain meta-models, making the system use domain-restrictive. In our case, DaaS providers can introduce new ontologies in the framework, thereafter new domain concepts are dynamically defined in the meta-model and can be used for the services annotation.

2.2.8.2 Service Composition

We recall that given a set of available data services and a user data query, the problem of service composition is concerned with synthesizing a new composite service that satisfies the data request, by suitably coordinating the available services. Our work is related to service composition in that we need to identify and compose the relevant data services available in the Service lake that can be used to populate the missing data and to fill the informational gaps in local data sources. Several approaches have been proposed for composing data services, which carry out complex interactions between the different services [73]. For each of the works, we discuss how they tackle such problem by focusing in particular on (1) how data queries of users are modeled, (2) the kind of composition, (3) the referred architecture for orchestration, and (4) based on which criteria, the pertinent data services are selected.

1) **S.A. Ghafour et al. [39]**

   In [39], S.A. Ghafour et al. presented an approach that caters for on-demand data integration for cloud business data needs. they presented an ontology-based semantic modeling for cloud DaaS services. DaaS Web services are modeled as RDF views over domain ontologies where primary key constraints are defined explicitly by the concepts skolem functions, thus the discussed Primary key based optimizations are included by default in their query processing model. An RDF view describes the semantics of a DaaS service in a declarative way using concepts and relations whose meanings are formally defined in domain ontologies. The proposed modeling makes it possible to automatically combine heterogeneous DaaS services and resolve the different types of data heterogeneity that would arise when data needs to be exchanged between composed services. Consequently, users need only to focus on the needed data by formulating their composition queries over domain ontologies. They are not required to manually select services and build the composition plan by mapping the inputs and outputs of component services to each other and drop code to resolve data incompatibilities.

2) **WSMED[72]**

   The Web Service Mediator System WSMED [72] allows users to mashup data services by defining relational views on top of them. It provides primitives for defining relational views of web service data and supports SQL queries over the views. Users can then query data by formulating their mashup queries over defined views. Users can also enhance defined views with primary-key constraints

to optimize the mashups. The main drawback of the WSMED system is its high reliance on users who are supposed to import services, define views and enhance the views with primary key constraints, thus requiring a good understanding of the services semantics.

3) **M. Barhamgi et al. [17]**

In [17], authors propose a new approach to automatically compose primitive data providing Web services for the purpose of creating data integration applications. Data providing services are modeled as RDF Parameterized Views over mediated ontologies. Then, an RDF oriented query rewriting algorithm is used to compose services for answering received queries. The composition is then optimized and deployed as a new Web service accessible on top of the Web.

4) **M. Arafati et al. [12]**

The paper [12] proposes a cloud-based DaaS framework to integrate private data from multiple DaaS providers with the goal of preserving both data privacy and the data mining quality of the underlying data. Authors assume that the data is shared in the form of a relational table, vertically partitioned into sub-tables, each of which is hosted by one DaaS provider. The data consumer submits a sequence of data queries to a mashup coordinator in the platform, where each query consists of the requested attributes, the required data quality (classification accuracy), and the maximum bid price. Since a single DaaS provider might not be able to provide all requested attributes, the mashup coordinator is responsible for determining the group of DaaS providers that can cover all the attributes while meeting the requested data quality and price. Finally, the mashup coordinator has to return an anonymized data table that satisfies a given privacy requirement that is agreed on by all the contributing DaaS providers.

5) **ANGIE [66]**

As we described earlier, ANGIE consists of enriching local knowledge bases from RESTful APIs and SOAP services by discovering, composing and invoking services to answer a user query. The authors assumes the existence of a global schema incorporating both data entities and definitions of data services. This assumption makes ANGIE domain specific and not suitable for general purpose queries. Furthermore, the focus being to compute the maximal number of answers, the proposed algorithm for the selection and the composition of relevant data services relies excessively on a depthfirst search strategy. This may involve

an infinite rewriting of the input query, and thus the algorithm will descend into a non-terminating recursion. As a primitive solution, the authors proposed to bound the depth of the derivation by MAX in order to prevent infinite loops.

6) **HYPATIA** [29, 28]

HYPATIA is a system for answering hybrid queries through accessing and processing data by coordinating both data services and computation services in dynamic environments. The notion of hybrid queries has been introduced in [29] to involve mobile and continuous queries and to be evaluated on the top of on demand or streaming static or nomad data services. Queries in HYPATIA are entered via a GUI and specified in a query language similar to CQL. Thereafter, evaluating an hybrid query consists of converting it into an executable form as a workflow, finding the relevant (data and computation) services, mapping each of the workflow activities to its corresponding service and finally ensuring the communication and inter-operation between the different activities. The workflow is represented as a directed graph whose vertices and edges denote the parallel and sequential composition of activities. Each activity in turn corresponds to a service call either for retrieving or processing data using data services or computation services, respectively.

7) **Academic Mashup Systems**

In other academic mashup systems [34][59][75][76], users are required to select the data services manually, figure out the execution plan of selected services and connect them to each other and drop JavaScript code to mediate between incompatible inputs/outputs of involved services.

Table 2.2 summarizes the different data service composition approaches described above and highlight the positioning of our composition approach among them.

2.2.9 Discussion

Most existing composition approaches [83, 17, 12, 39, 35, 75] assume that the data provided by different data services is complete, and thus, the answering query problem is essentially to identify all possible service compositions. Once the latter is solved, the optimal service composition is then selected to answer the data query. Nonetheless, this assumption is unrealistic as long as web services provide access to incomplete web sources. In fact, the data stored in these sources may overlap or complement each other, yet it is usually incomplete. Consequently, the selection of the optimal service composition, based on QoS values, may not completely satisfy the user data queries.

Composition Approach	Views Definition	QoS Factors Considered	Answers	Execution Cost
H. Elmeleegy et al. [35]	N/A	none	only a part	optimal
S.A. Ghafour et al. [39]	manually	none	only a part	optimal
J. Tatemura et al. [75]	N/A	none	only a part	optimal
L. Zeng [83]	N/A	response time, reliability, availability, reputation, price	only a part	optimal
WSMED[72]	semi-automatic	none	only a part	optimal
M. Barhamgi et al. [17]	manually	none	only a part	optimal
M. Arafati et al. [12]	N/A	price	only a part	optimal
ANGIE[66]	manually	response time	all answers	very high
HYPATIA[29, 28]	N/A	none	only a part	N/A
EuDaSL	automatic	response time, reliability, availability, reputation, price	almost all answers	high

N/A (Not Applicable): service views are not used in the underlying approach

Table 2.2 – Comparative View of Service Composition Approaches

Other recent efforts [66] aim to find the maximal number of answers to user queries by executing all candidate service compositions. This requires the execution of a huge number of web calls which, in turn, leads to a high execution cost (i.e., financial and computational costs). In spite of this, multiple service compositions with incomplete yet complementary results can eventually yield results that maximize user satisfaction. However, executing all possible compositions may not be necessary to get the requested data whenever executing some of them is sufficient to satisfy the need of the users.

Since we focus on computing the maximal number of answers without executing too many web calls, our mission is to find a compromise between the approaches described above and take the best from both of them. In such a context, we propose a new selection approach in which only a set of service compositions is selected based on a generic QoS model. Our goal is to compute the maximal results for user queries and to fill the eventual gaps of information found in their local data sources. Moreover, we make sure that we satisfy user requirements in terms of execution cost and service quality.

2.3 Concluding Remarks

We introduced in this chapter the background material and the state of the art focusing on the two major fields related to our work: data integration and service

composition. On the one hand, the proposed data integration approaches handle combining data from different and heterogeneous data sources so that they form a unified new whole and provide users the illusion of interacting with one single information system. Thereafter, users are provided with a homogeneous logical view of data that is physically distributed over heterogeneous data sources but not those data sources encapsulated by the service interface. On the other hand, a rapidly increasing suite of data services provide access to timely and high-quality information on the web. This makes Web services an interesting device for answering user data queries. However, neither data integration systems or service composition tools properly address the challenges raised by the continuous production of data and thereafter the lack of timely and up-to-date information in local data sources. With the *EuDaSL* system, we have shown that data services can step in to fill the informational gaps in companies' data sources. Given a user data query, EuDaSL enriches companies' datasets with the data it extracts on the fly using data services in order to provide complete answers to user queries. Service selection and composition are done in the background and data is seamlessly and transparently integrated in the local data source. Moreover, we propose *MoDaaS*, our model-driven framework for the modeling and the description of data services to cope with the heterogeneity regarding data services and the associated data assets. Our proposal to tackle this problem is based on Model Driven Engineering (MDE), a development paradigm exploiting the use of domain models to raise the level of abstraction and automation at which data services are developed. The next chapter will be devoted to introduce *EuDaSL*, our user-centric data integration approach. .

Chapter 3

User-Centric Data Integration in Service Lakes

Contents

In the light of the motivations expressed in the introduction chapter, we propose a complete solution to leverage the missing information in user datasets in order to be able to answer the queries he is interested in. To do so, we explore the possibility of acquiring the missing information by invoking DP web services on the fly. In this chapter, we present more details about our data integration approach and the overall architecture, giving the main steps required for answering a user data query. Then, we describe in a second part the two first steps of our integration process.

3.1 Data Model

Companies' data sources often employ heterogeneous data formats (e.g., text files, CSV files, XML documents, relational databases). Handling data and schemas from different models requires a common framework in which all the different data models can be presented. For this purpose, we uniformly represent the schemas and the data instances of such sources using graphs. Before proceeding to show how we do so, we start by introducing the data model we adopt in this work. More specifically, we distinguish between the schema level and the data level.

In the following, we consider the example dataset introduced above in Section 1.1 to illustrate the differences between the data graph and the schema graph.

Relation schema
Person(personID, first name, last name, date of birth, country) Author(authorID, name, university, email, domain) Book(iD,title, author,topic)
Foreign Keys
Table Author: authorID references personID of Person Table Book: author references authorID of Author

3.1.1 Schema Graph

The schema graph is a labeled directed graph G_S= (V, E), depicting the schema of the dataset, where `V` represents the different concepts, each one is characterized by a name and a set of attributes, and `E` is a set of labeled edges representing relationships between the nodes in `V`. We use `v.name` and `v.attributes` to denote the name and the attributes characterizing a node `v` $\in$ `V`, respectively. Similarly, we use `e.label` to denote the label of an edge `e` $\in$ `E`. If we consider a relational database, a node `v` $\in$ `V` would represent a relational table and `v.attributes` refers to the attributes of the relational table, while `E` represent referential integrity constraints between different tables in the database.

Once enriched, some of the nodes in the schema would refer to missing concepts that are populated using data services. Similarly, attributes in `v.attributes` represent the attributes that are associated with the concept.

Figure 3.1 illustrates how the relational schema presented above can be represented using our data model.

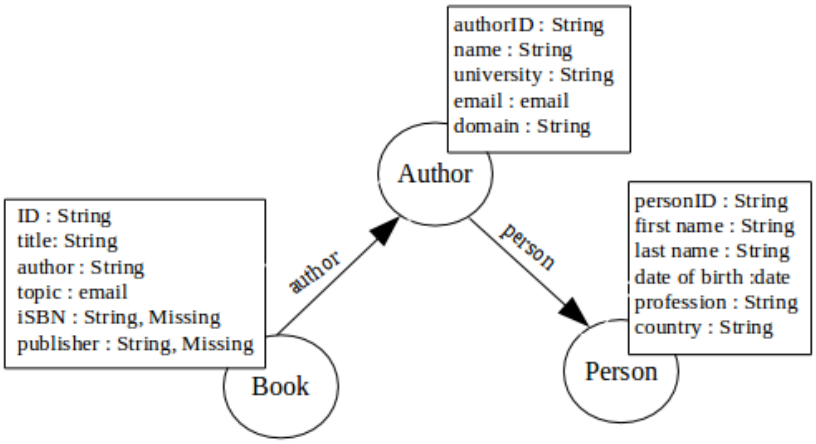


Figure 3.1 – An Example of a Schema Graph

3.1.2 Data Graph

The data graph is a directed graph G_D= (V', E', f_{ins}) where V' is a set of vertices representing the content (e.g., tuples in local data sources or the records retrieved by data services) of the dataset having the schema described in G_S and E' represents the relationships between the vertices in V'.

- Each node $\mathtt{v}' \in \mathtt{V}'$ represents an instance of a node $\mathtt{v} \in \mathtt{V}$ from $\mathtt{G_S}$. v' is characterized by a set of attribute-value pairs of the form <name, value>. The names that appear in those attribute-value pairs refers to v.attributes.
- The edges $\mathtt{E}'$ in the data graph $\mathtt{G_D}$ are used to enforce the constraints defined within the schema graph $\mathtt{G_S}$.
- $\mathtt{f_{ins}}(\mathtt{v}')$ is a function that given a node v' from the data graph, returns the node in the schema graph that represents the type of v'.

Figure 3.2 depicts a fragment of the data graph obtained by instantiating the schema graph in Figure 3.1. Nodes B1 and B2 represent two different books: the first written by both authors A1 and A2, and the second is written only by A3.

To be able to seamlessly treat concepts and attributes in the user data source, we use certain object oriented features such as we associate with each concept, considered as a class, an unary relation C(o) where o represents an instance of the concept C and a binary relation with each attribute Att(o, att) whenever att is the value of the attribute Att of the concept instance o. We use the convention that the relation associated with a concept has the same name as the concept, and similarly for attributes.

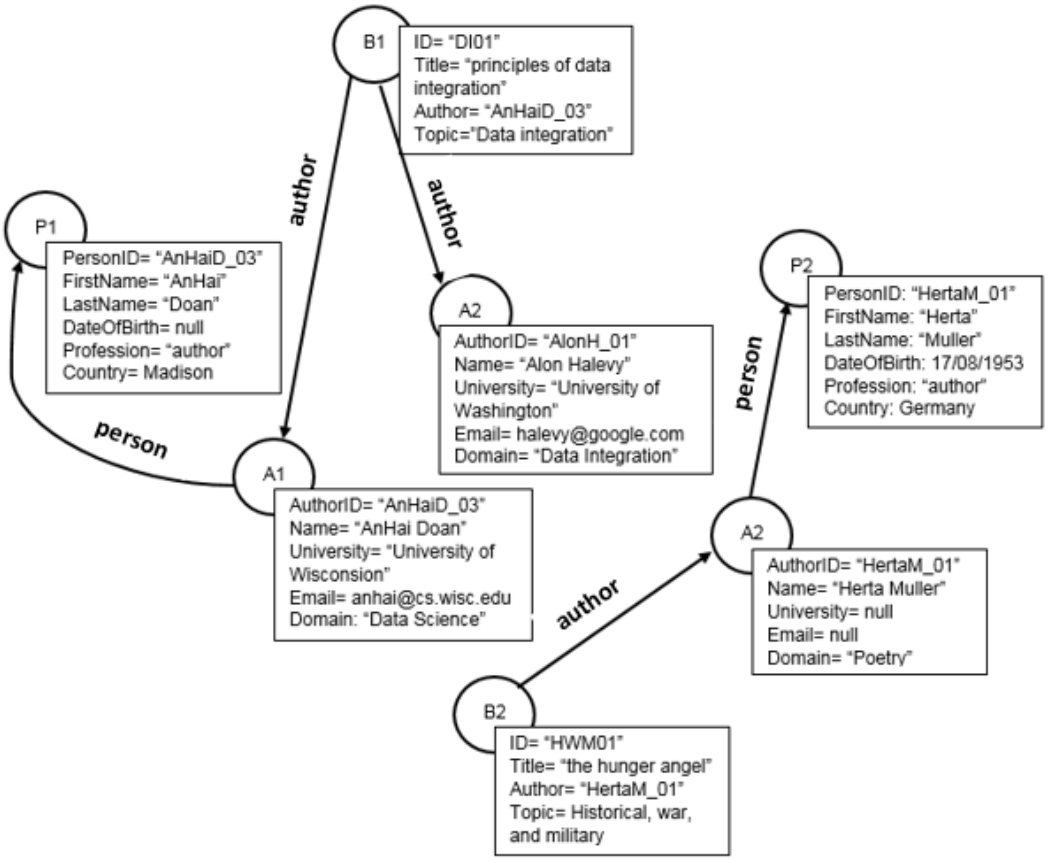


Figure 3.2 – A Fragment of the Data Graph Associated to the Schema Graph Introduced in Fig.3.1

Example 3.1.1 *Let consider the schema graph depicted in Figure 3.1, the concept Book is represented by the relation Book(o) such as o represents a specific book recorded in the database, let be the book 'principles of data integration', while the attributes id, title, isbn, etc., are represented by the binary relations id(o,i), title(o,t) and isbn(o,s), respectively.*

3.1.3 Data Queries

Users pose their data queries using an SQL-like query language syntax where the elements that are required by the queries but missing in the underlying dataset schema are prefixed with a question mark '?'. We consider the following data queries throughout this thesis:

Q_1 : `SELECT title, topic FROM Book.`

Q_2 : `SELECT ?iSBN, title FROM Book WHERE topic = 'Webservices'.`

Q_3 : `SELECT title, author, ?publisher FROM ?Publisher, Book WHERE ?Publisher .?name = Book.?publisher.`

Q_4 : `SELECT ?director, ?writers, ?stars FROM ?Movies.`

3.2 Data Provisioning Service Lakes

We introduce in this thesis a new paradigm that we call *data provisioning service lake* coined by analogy to data lake. A data provisioning service lake or the service lake for short is a storage repository of heterogeneous DP web services providing access to timely and high-quality information. The data returned by such services is retrieved from disparate web sources in its native format and stored in the raw data, as-is. The main idea behind service lakes is to take advantage of DP service capabilities in the lake while these services make data from web sources available through encapsulated APIs and to give minimal attention to creating schemas that define integration points between disparate provided datasets. Accordingly, instead of placing the retrieved data from different and

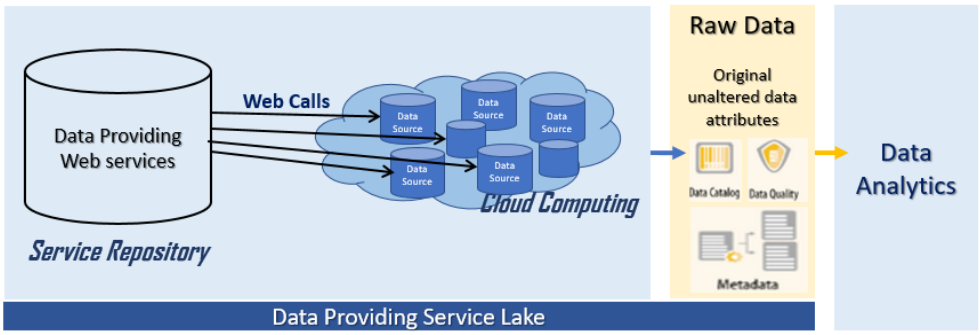


Figure 3.3 – Service Lake

heterogeneous web sources in a purpose-built data store, we move it into the lake, so that it may be later analyzed and mapped to the user data source schema. This facilitates and makes it possible to dynamically enrich user data sources for full query-answering purposes while eliminating the upfront costs and data ingestion.

3.3 Overall Architecture

Figure 3.4 presents the overall architecture of our data integration system which we call EuDaSL (***E***nriching ***U***ser ***DA***ta sources in ***S***ervice ***L***akes), briefly

sketches the purpose and responsibilities of each component, and highlights their interactions.

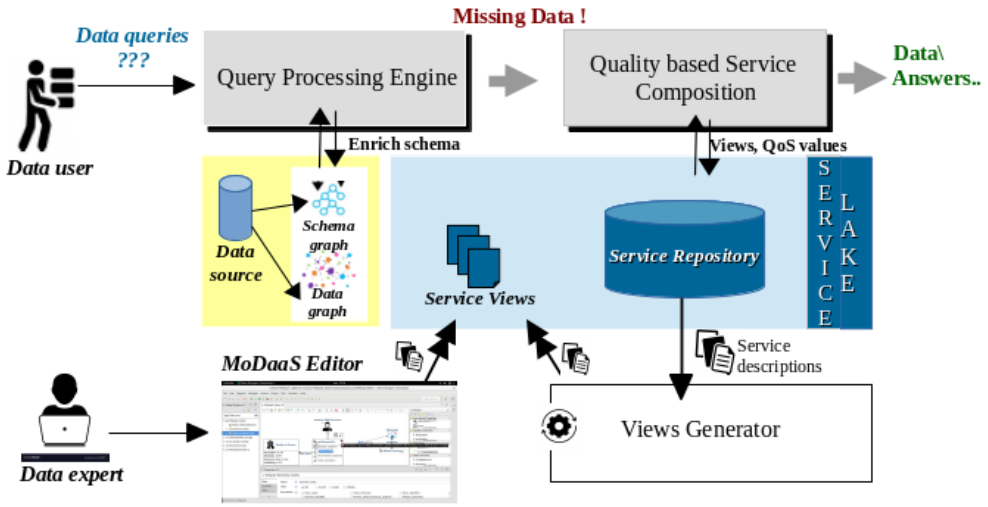


Figure 3.4 – EuDaSL Architecture

- The *Local Query Processing Engine* takes as input a user query, processes the schema graph of the local data source and determines the missing information that is required for the processing of the user queries. Thereafter, the determined missing concepts are defined in the schema graph as new nodes and missing relations are represented by edges, while the missing attributes have to be defined within the set of attributes of the corresponding concept.
- The *Data Service Composition Engine* translates the query into a sequence of service views compositions. These service views can be (semi-)automatically generated either by the *Service Views Generator* or within the *MoDaaS framework*. This is an important preliminary step to deal with the possible data heterogeneities that can exist between the user data query and the different data services in the Service Lake.
- The *Service Views Generator* automatically defines the top-k best views that map the capabilities description of data services into the schema elements of the user data source. The generated views are subject to user feedback for confirmation or correction.
- The *MoDaaS framework*, a Model-Driven framework for the modeling and the description of data services and DaaS services in particular. It assists

data experts and data providers to define and describe their services' capabilities according to a domain ontology and to automatically generate the corresponding views.

Thereafter, the generated views can be easily and automatically used by the *Service Composition Engine* for the selection and the composition of the relevant services that can be used to fill the informational gaps in the user data source. Furthermore, we implemented other components/tools that we did not present in the figure for the clarity sake:

- A *Graph Generator* to convert the user data source into a common graph-based model. More specifically, it extracts the schema graph and the data graph of the user data source.
- A *Query Rewriting Module* to reformulate the user data query as a conjunctive query over the relations in the schema graph. Once reformulated, some of the requested data in the initial query are represented by unary relations bound to constant values. These values are extracted from the local data source.
- A *Mapping Tool* to execute the Web service calls. It mediates between the schema of the XML and the Json documents that the service call returns and the relations in the schema graph. In particular, it defines how the XML/JSON nodes in the answer are mapped to concepts in the local data source. We use the XSLT standard [21] for the parsing of XML files and the Python Json module to parse JSON files. The values can then be seamlessly integrated in the local data source.

3.4 Query-Answering Process

Figure 3.5 illustrates the overall process of our data integration approach once service views are defined either automatically using the proposed solutions as shown in the previous section or manually by experts. Given a set of queries that are of interest to the user, and given a local dataset that is provided by the user, our solution proceeds as follows:

- Step1 determines the missing information that is required to process user queries but is not provided by the local dataset. This consists mainly on browsing the schema of the local dataset and deduce the missing concepts and/or relations.

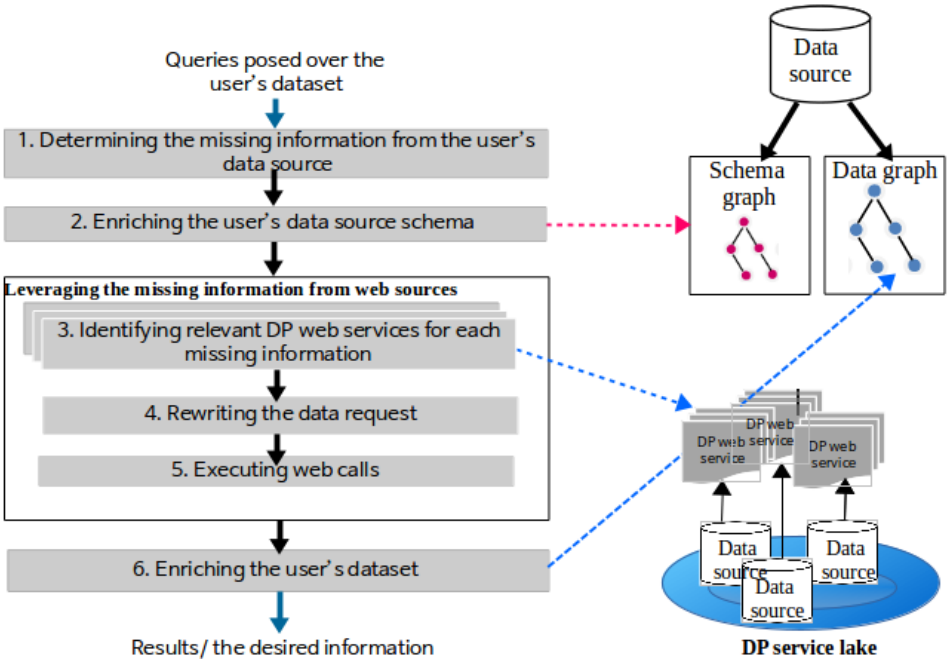


Figure 3.5 – An Overview of the Integration Process

- Step 2 enriches the schema of the local dataset by defining missing elements (concepts and/or attributes) determined in the previous step.
- Step 3 identifies the set of candidate data services in the lake that can be used to populate the missing information.
- Step 4 reformulates the user query over the relevant services views and selects the executable query plans satisfying user requirements in terms of data quality and cost.
- Step 5 evaluates executable query plans, which involves the call of DP services in the lake.
- Step 6 integrates the data obtained from web services into the local data source.

During the integration process, the data developer is expected to interact only to confirm or modify the system proposition for the enrichment of the schema of the local data source. In the following section, we focus on the first of the challenges mentioned above. Specifically, we show how the schema of a local dataset can be enriched with new concepts given a workload specifying the user queries.

3.5 On Enriching Local Data Sources' Schemas in Service Lakes

This section is structured as follows: in a first section, we present our algorithm for the determination of the missing information: the missing concepts, attributes and relationships. In the section thereafter, we describe our algorithm for the enrichment of the schema of user data source. In Section 3.5.3, we showcase the working of our solution using a study case.

3.5.1 Identifying Missing Information

In our model, the user can query for data that is not yet in his local data source and that will be later leveraged from web sources on the fly. The aim of this step is to identify the missing data, more specifically the missing concepts and associated attributes that are required by the user's queries but that are not provided by his data source.

Given a set of user queries $Q_D = \{Q_1 , \ldots , Q_n\}$ and the graph G_S representing the schema of the user data source, the algorithm 1 processes queries one by one with respect to the order specified in the workload. We consider in this work SPJ (select-project-join) queries , which we represent using the triple (attributes, concepts, conditions) in order to improve readability. Such triples specify respectively the set of attributes in the select clause of the query, the set of concepts involved, and the set of conditions that appear in the where clause of the query. The algorithm parses the queries and outputs the missing concepts, MissConcepts, as well as the missing attributes, MissAttributes, characterizing existing or new concepts. A missing concept is identified with a name: MissConcepts $= \{c_1 , c_2 , ..., c_n \}$. As we will discuss later on, it is not always possible to identify with certainty which concept(s) a given attribute characterize. Because of this, we define a missing attribute by the triple (name, concepts, certitude), where name is the name of the attribute, concepts represents the set of concepts to which the attribute belongs, and certitude is a variable that takes the value 'Certain' or 'Uncertain'. If a concept in the query is not represented by a node in the schema graph, the algorithm checks if this latter was already defined as a missing concept previously (line 7, Algorithm 1). If not, it defines it as a missing concept. If the concept is represented in the schema graph, the algorithm proceeds by verifying the existence of attributes related to it (lines 4-6, Algorithm 1). A missing attribute is an attribute that does not figure in the list of attributes defined within that concept in the schema graph G_S . As mentioned earlier, it is

not always possible to identify which concepts a given attribute belongs to. This is particularly the case for join queries. For this kind of queries, an attribute may characterize a subset of concepts that are involved in the query. In order to determine the appropriate subset, we evaluate the semantic relatedness score between a given attribute and each concept from the entire set of concepts stated in the query, based on information retrieved from external sources of knowledge. In our case, we make use of the lexical database WordNet [24].

Algorithme 6 : Searching for missing information

Require : Q_D= Q_1, Q_2, ..., Q_n is a data request, G_S= (V,E) is a schema graph
Ensure : MissElts= MissConcepts, MissAttributes, MissRelations
1 $MissConcepts \leftarrow \emptyset$, $MissAttributes \leftarrow \emptyset$, $MissRelations \leftarrow \emptyset$
2 **ForEach** Q_i *in* Q_D **do**
3 | **if** Q_i.*concepts involves only one concept c* **then**
4 | | **if** *there is a node* $v \in V$ *that corresponds to c* **then**
5 | | | **ForEach** Q_i *in* Q_D **do**
6 | | | | attribute att $\in Q_i$.*attributes* that does not belong to v.attributes
7 | | | add (att, c, 'certain') to MissAttributes
8 | | **end**

Definition 3 *The relatedness score estimates the degree by which two words are semantically related, which is a number between 0 and 1 [64].*

In the literature, several measure methods have been proposed to compute the semantic relatedness between two different words. In [63], the authors used the relatedness scores from the human studies; the *Miller and Charles(M&C)* study as well as the *Rubenstein and Goodenough(R&G)* research to evaluate six different relatedness measures. In this studies, human subjects assigned relatedness scores to different sets of word pairs ranged from highly related pairs to unrelated pairs. The derived scores have been used in [63] to evaluate and compare the selected measures, more specifically its correspondence with the human perception of relatedness. Table 3.1 summarizes the correlation coefficient of the ranking of each measure with that of the human relatedness. Notice that the **Gloss Vector** has always the highest correlation with human judgments according to the performed experiments (0.91 on *M&C* and 0.90 on *R&G*), we rely in our work on the **Gloss Vector** measure method for the computation of the relatedness score.

Miller and Charles: represents a dataset of 30 word-pairs rated by a group of 38 human subjects. The word pairs are rated on a scale from 0 to 4 such as the higher the number is, the higher the similarity is [56].

```
7
8
9       else if c was already defined in MissConcepts then
10        foreach attribute att ∈ Q_i.attributes do
11          if (att, concepts, certitude) was defined in MissAttributes such as
              c ∈ concepts and certitude='uncertain' then
12            replace (att, concepts, uncertain) by (att, c, 'certain')
13          else if att was not defined in MissAttributes then
14            add (att, c, 'certain') to MissAttributes
15      else add c to MissConcepts and all attributes in Q_i.attributes to
          MissAttributes;
16    else foreach concept c ∈ Q_i.concepts that does not have any representative
        node v ∈ V do
17      add c to MissConcepts
18    foreach condition cond ∈ Q_i.conditions do
19      get related attribute-concept pairs < att, c > in cond
20      foreach < att, c > such as (c is not missing from G_S and att is missing)
          or (c was defined as a missing concept however att was not defined as a
          missing attribute related to c) do
21        add (att,c,'certain') to MissAttributes
22      foreach linked concepts c1 and c2 in cond that are not related in G_S by
          an edge e ∈ E do
23        define a new edge e= (c1, c2, att1) in MissRelations where c1
            represents the outgoing node, c2 is the incoming node and att1 is
            the label of this edge
24    foreach a in Q_i.attributes that does not belong to any c. attributes and was
        not defined as a missing attribute related to c such as c ∈ Q_i.concepts do
25      concepts ← ∅
26      foreach c in Q_i.concepts do
27        compute the relatedness score("a", "c")
28        if c has a relatedness score higher than 0.5 then
29          add c to concepts
30      add(a, concepts, 'uncertain') to MissAttributes
31    ;
```

Rubenstein and Goodenough(R&G): is a dataset of 65 word-pairs rated by 51 subjects according to a scale from 0 (zero similarity) to 4 (total/perfect similarity)[71].

The Gloss Vector Measure: is a second order context vector formed by treating the dictionary definition of a concept as a context, and finding the resultant of the first order context vectors of the words in the definition [63]. This

Relatedness Measure	M & C	R & G
Gloss Vector	0.91	0.90
Extended Gloss Overlaps	0.81	0.83
Jiang & Conrath	0.73	0.75
Resnik	0.72	0.72
Lin	0.70	0.72
Leacock & Chodorow	0.74	0.77

Table 3.1 – Correlation to Human Perception [63]

measure joins together both ideas of concepts definitions from a thesaurus and co-occurrence data from a corpus. Every word in the definition of the concept from the dictionary (e.g., WordNet) is replaced by its context vector from the co-occurrence data from a corpus and then, relatedness is calculated as the cosine of the angle between the two input concepts associated vectors. This Gloss Vector measure is highly valuable as it: 1) employs empirical knowledge implicit in a corpus of data, 2) avoids the direct matching problem, and 3) has no need for an underlying structure

Given an attribute a_i and k concepts $c_1, \ldots, c_k$, we select the concepts having a relatedness score with the attribute in question a_i higher than 0.5. Otherwise, if any of the resulting relatedness score is higher than 0.5, we associate the attribute with the concepts having the highest score. As an example, we compute the relatedness score of 'book' and 'actor' versus 'actor' and 'movie', which yield to 0.29 and 0.41, respectively. In this case, our algorithm adds the attribute 'actor' to the concept 'movie'. Note that this reduces but does not eliminate the uncertainty about the selected concepts, since only humans are able to quickly judge the relative semantic relatedness of pairs of concepts. Because of this, our algorithm tags the missing attribute with an 'uncertain' certitude label. Thus, our method has the merit of reducing the number of concepts that needs to be examined by the user.

Furthermore, the fact that we process a workload of queries may help reduce the uncertainty. Though after a given query there may be uncertainty about the membership of a given attribute to a given concept, another query may confirm it. In such a case, we update the certainty tag of the attribute in question from 'uncertain' to 'certain' (lines 8-9, Algorithm 1).

Note that even with the knowledge base WordNet, attributes to be integrated could not be determined exactly due to the complexity in semantics. We thus allow an interactive and iterative enrichment process during which the user can

provide his feedback, e.g. to manually define new attributes or concepts to the data source schema, or to confirm or reject the proposed modifications if he is not entirely satisfied by the system proposition.

3.5.2 Enriching Local Data Sources

If the execution of algorithm 1 leads to a non-empty set of missing elements, the system must update the user data source at the schema level as well as at the data instances level, as described in the following paragraphs.

Schema Level: Enriching the schema graph consists in adding new concepts and attributes to those already defined in the source schema. Algorithm 2 creates for each missing concept in MissConcepts a new node in G S and adds missing attributes to the corresponding concepts listed in attribute.concepts (lines 1-5, Algorithm 2). In the following, we differentiate missing elements in the graph by the label 'Missing'. The algorithm also defines new integrity constraints/semantic relations between different nodes of the graph (lines 6-7, Algorithm 2).

Data Instances Level: Unlike schema enrichment which comes immediately after the identification of missing concepts and attributes, data graph enrichment can only be performed once data services are invoked and the missing data is retrieved by them. That is why mapping generation between web service call results and the concepts in the schema graph is addressed later in this thesis. In fact, the results returned by a data service call are used to populate the concepts and associated attributes in the data graph of the local data source. Once the system

Algorithme 7 : Enriching Schema Graph

Require : $G_S = (V, E)$, MissElts= MissConcepts, MissAttributes, MissRelations
Ensure : G_S (the enriched Schema Graph)

```
1 foreach c ∈ MissConcepts do
2     add a new node named c with the label 'M' to V
3 foreach att ∈ MissAttributes do
4     foreach c ∈ attribute.concepts do
5         define a new attribute att additionally to initial attributes defined within
            the concept c, having 'String' as a type and 'Missing' as a state
6 foreach rel ∈ MissRelations do
7     add a new edge e to E outgoing from rel.OutNode, incoming to rel.InNode
        and labeled with rel.label
```

has finished the evaluation of all the queries in the workload and enriched the

local data source, it removes all 'missing' tags from the schema graph, thereby preparing the environment for future user interrogations.

3.5.3 Case Study

Continuing with the relational database introduced in the motivating scenario (cf. Figure 3.1), we illustrate in this section how algorithms 1 and 2 respectively operate on a sequence of three queries: Q_2, then Q_3, and finally Q_4. Q_2 and Q_3 are defined above in Section 3 of Chapter 3 and we will define Q_4 below.

Q_2 involves only the relational table 'Book', that we consider as a concept in our model, in order to get titles and iSBNs of all books stored in the table. Algorithm 6 first searches for iSBN in the list of attributes of 'Book'. It does not find it, therefore it defines it as a missing attribute (iSBN, Book, certain) in MissAttributes. Then, it proceeds by processing the next query in the workload, Q_3. Algorithm 6 first verifies the existence of all the concepts involved in Q_3 in the schema graph G_S, concluding that Publisher is not represented in G_S . It also examines $Q_3.conditions$ to identify that an integrity constraint between the relational tables 'Book' and 'Publisher' was not be represented in the initial schema, and that the missing attribute 'name' must be added within the concept 'Publisher', whereas 'publisher' should be defined additionally to Book.attributes.
Now we apply algorithm 6 to Q_4: All requested information in Q_4 is not represented in the database, be they relational tables ('Movies') or attributes. As a consequence, algorithm 6 defines Movies as a missing concept, director, writers and stars as missing attributes.

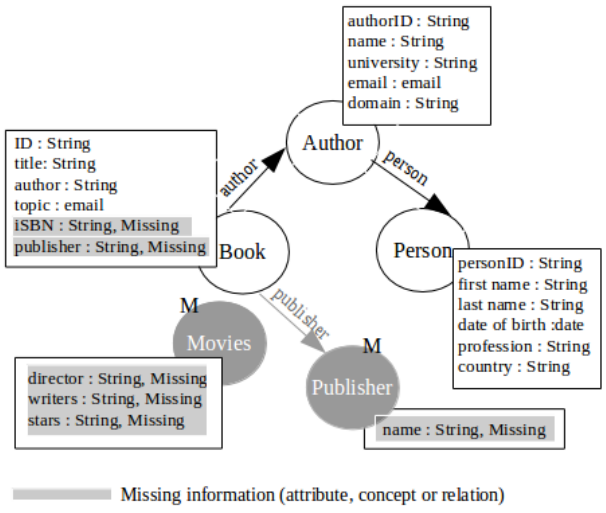


Figure 3.6 – An Enrichment Example of the Schema Graph Introduced in Fig.3.1

In the second step, algorithm 7 updates the schema graph by representing missing elements defined earlier by algorithm 6 in MissElts. As explained in Section 3.5.2, the representation of missing concepts is done before the definition of missing attributes. Therefore, two new nodes representing respectively the relational tables 'Movies' and 'Publisher' are added to the schema graph, labeled with the character 'M' to denote that they basically represent missing concepts. Then, iSBN and publisher are added to the set of attributes Table.attributes and finally all of the attributes director, writers and stars are defined as missing attributes characterizing the concept 'Movies'. Furthermore, the algorithm creates a new integrity constraint 'publisher' between the relations 'Book' and 'Publisher'.

All of these modifications are depicted in Figure 3.6: nodes and relations in gray represent respectively missing concepts and missing relations, and the attributes with a gray background represent the missing attributes.

Consider a schema graph of n nodes, and a data query requesting m attributes related to at most l concepts and under q conditions. Algorithm 6 runs in $\mathcal{O}(m.l.n)$ time, while Algorithm 7 clearly runs in $\mathcal{O}(m.l)$ time.

3.6 Concluding Remarks

We introduced in this chapter the overall architecture of EuDaSL. Then, we provided a step-by-step explanation of the process of our data integration approach. EuDaSL aims to enrich the user data source with information extracted from web sources whenever the local dataset does not suffice to answer the user queries, w.r.t user requirements. This process is not entirely automatic such it relies on a database designer/developer decisions for the final choice of service views. Then, the work presented in the second part of the chapter tackles the determination of missing information in local data sources and the enrichment of the schema with the ability to define new concepts to the user data source schema. The missing concepts are deduced from the user data queries but are not defined in the initial schema of the user datasets. In the following chapter, we present our approach for the leveraging of the missing data using data services in the service lake.

Chapter 4

Quality-based Data Service Composition for Enriching User Data Sources

Contents

In the previous chapter, We have shown how local data source schemas can be enriched with new concepts that are required for the processing of a given collection of data queries. In this chapter, we go further in this direction and show how such queries can be evaluated through combining data coming from local datasets and data provided by data services. More specifically, our aim

is to select the relevant data services that yield good quality answers without exceeding a given budget set by the user (time and monetary cost) and seamlessly enrich the local data source with the obtained answers. Thereafter, the local data source may be leveraged to obtain complete answers satisfying user queries. This is performed through the steps 3, 4, 5 and 6 of our data integration process, presented in Section 3.4 of Chapter 3.

This chapter is organized as follows: In Section 4.1 , we first introduce the notations that we adopt for the composition of data services. Next, in Section 4.2, we present the algorithms that we developed for the identification and the composition of relevant data services. In Section 4.3, we describe our algorithm for the selection and the execution of the best quality query plans while satisfying the user constraints. Then we evaluate the Composition framework from different aspects as a whole and on the elementary levels separately. Some of the results presented in Section 4.4 of this chapter were published in the conference *IEEE International Conference on Web Services (IEEE ICWS 2018)* [7]. Finally, we conclude in Section 4.5.

4.1 Preliminary Steps

In this section, we first present the pre-processing steps to perform by the designer or simply the data developer before starting the use of our system. This consists of defining the data service views over the relations in the schema of the user data source. Then, we describe the query language that we use for the rewriting of the user data query and for the composition of relevant service views.

4.1.1 Data queries

User queries (or data queries) are issued against the schema graph of the local data source. Specifically, a data query is reformulated as a conjunction of unary and binary relationships on the concepts and attributes in the schema graph, respectively.

$$\mathtt{Q}(\overline{\mathtt{X}}) \rightarrow \mathtt{q}_1(\overline{\mathtt{X}}_1), \ldots, \mathtt{q}_m(\overline{\mathtt{X}}_m), \mathtt{C}_I \tag{4.1}$$

where $q_1, ..., q_n$ are the relations representing concepts and attributes in the schema graph and C_I represents integrity constraints between these different concepts. Integrity constraints provide a way of ensuring that changes made to the database do not result in a loss of data consistency. For example, the following query asks for titles, authors' names and publication date of all the books

recorded in the library's database:

$$\mathtt{Q}(\mathtt{t},\mathtt{p},\mathtt{a},\mathtt{p}) \leftarrow \mathtt{PDF\_Books}(\mathtt{o_b}),\mathtt{Book\_Title}(\mathtt{o_b},\mathtt{t}),\mathtt{Author\_Names}(\mathtt{o_b},\mathtt{a}),\\ \mathtt{Date\_of\_Publication}(\mathtt{o_b},\mathtt{p})$$

Not all the concepts used in the user query will have instances associated with them in the user data source (missing concepts). In the subsequent section, we show how we use service views to reformulate users' queries in a way to replace the missing concepts with sub-queries that retrieve data from the services.

4.1.2 Service Lake

Queries are posed to the system in terms of the concepts in the data source of interest. However, some of the requested data in the data query can be missing in the local data source. In this work, we investigate the Service Lake to obtain the missing data from external web data sources, using the data services.

A service S is described by $S = (In_S\ ,\ Out_S\ )$ where In_S is the set of input parameters needed for the execution of the service s and Out_S is the set of output parameters provided by the service. A service description can be in any known Semantic Web Services formalism (often abbreviated with SWS) that is RDF/OWL based and that describes the functional and the non-functional features of a service.

To be able to answer a query posed over the user data source, we need to describe the capabilities of the data services in the Service Lake in terms of the concepts and relations in the local data source. We first define schema mappings, that we termed service views, between the schema graph of the data source and the services in the lake.

4.1.3 Data Service Views

In this work, a data service view captures in a declarative way the relationships between input and output parameters of the data service in question and the concepts and the relationships in the schema graph of the local data source. Service views are modeled as conjunctive queries over the relations in the schema graph. Let consider the schema graph depicted in Figure 4.1 which corresponds to a library database.
For sake of simplicity, we represent only the table of interest from the database. Following our model, the concept PDF_Books is represented by the relation

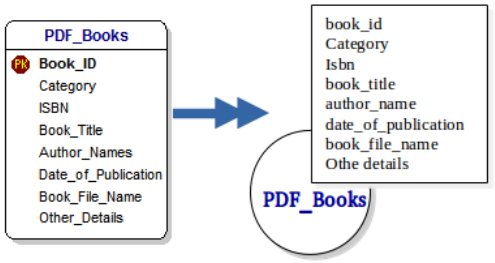


Figure 4.1 – An Excerpt of a Library Database Schema & Its Corresponding Schema Graph

PDF_Books(o) and the attributes book_id, category, isbn, etc., are represented by the binary relations book_id(o,i), category(o,c) and isbn(o,s), respectively.

Definition 4 *Conjunctive Queries: A conjunctive query is a restricted form of first-order queries using the logical conjunction operator. It is simply the fragment of (domain independent) first-order logic given by the set of formulae that can be constructed from atomic formulae using conjunction $\wedge$ and existential quantification $\exists$, but not using disjunction $\vee$, negation $\neg$, or universal quantification $\forall$. Syntactically, a conjunctive query q is an expression of the form:*

$$q(\overline{x_1}, \ldots, \overline{x_n}) : \neg R_1(\overline{y_1}), \ldots, R_n(\overline{y_n}) \tag{4.2}$$

The expression $q(\overline{x}_1, \ldots, \overline{x}_n)$ is called the head of the query, and $R_1(\overline{y}_1), \ldots, R_n(\overline{y}_n)$ is called the body of the query, where $n \geq 0$, R_i is called an atom and q is a fresh relation name. The expressions $\overline{x}_1, \ldots, \overline{x}_n, \overline{y}_1, \ldots, \overline{y}_n$ are called free tuples, and contain either variables or constants [36].

Let us consider the data service ***getAuthor*** which given a book ***title*** and its ***isbn*** returns the ***name*** of the book author:

$$\{\texttt{getAuthor}, \{\texttt{title}, \texttt{isbn}\}, \{\texttt{author}\}\}$$

The corresponding view among the library database is defined as conjunctive query over the relations in the database schema as follows:

$$\texttt{v}_{\texttt{getAuthor}}(\texttt{t}, \texttt{isbn}, \texttt{a}) \subseteq \texttt{PDF\_Book}(\texttt{o}_1), \texttt{book\_title}(\texttt{o}_1, \texttt{t}), \texttt{isbn}(\texttt{o}_1, \texttt{isbn}), \texttt{author\_name}(\texttt{o}_1, \texttt{a})$$

It involves the attributes book_title, isbn and author_name of the concept PDF_Book. o_1 is the identifier of the object/tuple in the library database

while t and isbn represent variables that should be bound to values either provided by the user in the query, obtained from the library database, or by the output of other data services in the context of a service composition before the query is executed. Thereafter, the answer of the query associates binding values for the variable a. Note that at this level, we do not distinguish between input and output parameters within the service view.
Actually, we adopt the Local as View (LAV) integration approach by describing which concepts of the user's data source can be leveraged by data services.
Continuing with the service view $V_{getAuthor}$, the corresponding operation can be used to populate the attribute `author_name` of books `PDF_Books` recorded in the library database (cf. Figure4.1).

4.2 Selection and Composition of Data Services

Given a user query, the problem is to evaluate it using the local data source and the set of data services in the data provisioning service lake.

Individual services may provide only some data and not all the missing information, therefore answering a data query may require investigating different combinations of different data services, which we also call service compositions.
However, data queries have to follow the binding patterns of service operations, by providing values for required input parameters before the operation can be called. For example, the service operation *getAuthor* can be executed only if a book title and its ISBN are provided, however, if the query requests for the titles of books written by a given author, the operation cannot be executed to answer this question, even though the database contains the desired information. Because of this restriction, we can find non-executable compositions, then an adequate orchestration between different component services is necessary to look for an executable ordering, if it is possible. The ordering set making a composition executable is called a query plan.

The construction of executable compositions comprises two phases. In the first, we create all the possible service compositions that can be used to leverage the missing information and fully satisfy the user queries. In the second, we try to order the conjuncts of the compositions in order to ensure that they can be executed and we prune away non executable plans. We provide more details in the two next subsections.

4.2.1 Composition of Pertinent Data Services

Given a query Q and a set of data services represented by their corresponding views $V = v_1, v_2, ..., v_i$, the first step consists of identifying all service views containing all or a part from the conjuncts constituting the data query.

First of all, Algorithm 8 makes sure that the information present in the local database is used first to obtain the available information. To do so, the algorithm checks for every conjunct $\mathtt{q_i}(\overline{\mathtt{X}}_\mathtt{i})$ if it represents a missing concept or not. If it does not represent a missing concept, the algorithm determines all possible mappings between the variables stated in conjuncts and those ones stated in relations. However, if the concept is denoted as missing, the algorithm proceeds by checking the set of DP web services views, SV, to get it in order to fill the gap on the fly. Relevant services must provide the missing attribute, otherwise, it is useless to call them. For each relevant service view sv, the algorithm determines all the mappings and add $\mathcal{M}(\mathtt{sv})$ to $\mathtt{RelevantViews_i}$.
If the conjunct $\mathtt{q_i}(\overline{\mathtt{X}}_\mathtt{i})$ is denoted as a missing concept, there is no need to check local views (views provided by local databases), the algorithm starts directly by determining the relevant service views to answer the data query.

In the second step, the algorithm checks the cartesian product of the buckets. We consider conjunctive plans that are obtained by selecting one relevant service views for every conjunct in the query $\mathtt{Q}$. The generated plans are guaranteed to be sounds and relevant. Indeed, unlike the work proposed by Levy et al. [49], which considers order relations $\{\leq, \leq, =, \neq\}$, we do not consider these relations since they are generally not used to describe service operations. As a such, the plans generated using our algorithm are guaranteed to be answers to the user query, and thus we have soundness. Moreover, we do not consider disjointedness[2] relationships between the concepts in our schema graph. Therefore, the query plans generated may always produce answers, hence relevance.

4.2.2 Creation of Executable Query Plans

Data services allow querying remote databases. However, the data request has to follow the binding patterns of the Web service operations, by providing values for mandatory input parameters before the operation can be called. Given the specification of the composite service synthesized in the previous phase, we need to

[2]A constraint about generalization hierarchies that addresses the question whether an instance of a supertype may simultaneously be a member of two (or more subtypes)

Algorithme 8 : Identify Relevant Service Views

Require : $\mathtt{Q}(\overline{\mathtt{X}}) \rightarrow \mathtt{q}_1(\overline{\mathtt{X}}_1), \ldots, \mathtt{q}_m(\overline{\mathtt{X}}_m), \mathtt{C}_I$ is a conjunctive query
$\mathtt{SV}$ a set of service views
Ensure : $\{\mathtt{RelevantViews}_1, \ldots, \mathtt{RelevantViews}_m\}$

1 **for** *every conjunct* $\mathtt{q}_i$ *in* $\mathtt{Q}$ **do**
2 $\mathtt{RelevantViews}_i \leftarrow \emptyset$
3 **if** $\mathtt{q}_i(\overline{\mathtt{X}}_i)$ *is not missing* **then**
4 there exists a relation $\mathtt{R}(\overline{\mathtt{Y}})$ that corresponds to $\mathtt{q}_i(\overline{\mathtt{X}}_i)$
5 let $\mathcal{M}$ be the mapping defined on the variables of $\mathtt{R}(\overline{\mathtt{Y}})$ as follows: **if** $\mathtt{Y}$ *is the j'th variable of* $\mathtt{R}$ **then**
6 $\mathcal{M}(\mathtt{Y}) \leftarrow \mathtt{X}_j$ where $\mathtt{X}_j$ is the j'th variable in $\overline{\mathtt{X}}_i$
7 add $\mathcal{M}(\mathtt{R})$ to $\mathtt{RelevantViews}_i$
8 **else**
9 **for** *every conjunct* $\mathtt{u}(\overline{\mathtt{Y}})$ *in a body of a service view* $\mathtt{sv}$ *in* $\mathtt{SV}$ **do**
10 **if** $\mathtt{q}_i = \mathtt{u}$ **then**
11 let $\mathcal{M}$ be the mapping defined on the variables of $\mathtt{sv}$ as follows: **if** $\mathtt{Y}$ *is the j'th variable of* $\mathtt{u}$ *and* $\mathtt{Y}$ *is not existential* **then**
12 $\mathcal{M}(\mathtt{Y}) \leftarrow \mathtt{X}_j$ where $\mathtt{X}_j$ is the j'th variable in $\overline{\mathtt{X}}_i$
13 **else**
14 $\mathcal{M}(\mathtt{Y})$ is a new variable that does not appear in $\mathtt{Q}$ or $\mathtt{sv}$
15 add $\mathcal{M}(\mathtt{sv})$ to $\mathtt{RelevantViews}_i$

coordinate the various component services, and monitoring control and data flow among them, in order to guarantee the correct execution of the composite service. The resulting ordering is called "Query Plan". In this section, we show how we create executable query plans from a service composition (i.e., plans whose all composing services can be executed while its required inputs are available, some of the inputs can be obtained on-the-fly after the execution of previous services in the composition).

Algorithm 9 inspired from [50] tries to find an executable ordering on the component services of a composition so that the plan becomes executable, if such ordering exists. In [50], the authors proposed an algorithm for computing an executable ordering of a set of services each one performing a specific task. The algorithm proceeds by maintaining a list of available parameters, and at every point adds to the ordering any sub-goal whose input requirements are satisfied. In our work, we make sure that each service operation involved in the query plan has its input parameters satisfied. Those are bound to values that are provided

by the user in the query, by the concepts that are not missing in the generated plan, or by the output of service operations that are previously called within the query plan. We notice that a plan may not have an executable ordering on the service operations it involves. In such a case, the plan in question is not considered for answering the user query.

4.3 Quality based Selection of Query Plans

Filling the gaps in the user's data source involves some challenges such as identifying relevant data providers meeting the consumer's data request requirement, the data quality requirement, the bid price as well as getting as many results as possible. Certainly, some data concerns can be assessed before the data is exposed through data services operations. Such information known as QoS (Quality of Service) associated with data services can help deciding whether a data resource should be used and under which conditions. In this step, we aim to ensure the selection and the execution of query plans that not only satisfies the data query but which also have together the best quality.

Algorithme 9 : Create Executable Query Plans

Require : a conjunctive plan $\mathtt{Q}$‘ corresponding to a possible combination of relevant views $(\mathtt{R_1}, ..., \mathtt{R_l}, \mathtt{V_1}, ..., \mathtt{V_m})$
I/O parameters of a data service V_i is $(\mathtt{in_i}, \mathtt{out_i})$

Ensure : a query plan $\mathtt{P}$

1 $\mathtt{BindAvail_0}$ = The set of variables in Q‘ bound by values in the query and those returned by local relations $R_i, i \in [1..l]$
2 Q_{out} = The head variables of Q‘ (the same head variables of Q).
3 $j = 2$
4 **while** *there exists also service views that were not chosen earlier* **do**
5 **foreach** V_i *in* Q **do**
6 **if** V_i *was not chosen earlier and the parameters in* in_i *are in* $\mathtt{BindAvail_{i\_1}}$ **then**
7 V_i is the j‘th service in the ordering
8 $BindAvail_i = BindAvail_{i\_1} \cup out_i$
9 $j \leftarrow j + 1$
10 **if** $Q_{out} \not\subseteq BindAvail_m$ **then**
11 plan is not executable

4.3.1 Measuring QoS values of a Service Composition

This step consists of evaluating the QoS values of composite service executions such as the execution cost, the service reputation, its availability, its reliability and the estimated response time. The QoS of a resulting query plan is a determinant factor to ensure the quality of returned answers and the user's satisfaction. The operation that has the largest impact on the total execution time and the execution cost is the execution of web calls. So our algorithm should prioritize the execution of the calls in web call composition that lead to a solution with a smaller cost (i.e., minimize the response time and the financial cost) while maximizing the services reputation and availability, etc.
Estimating the quality of a service composition is not straightforward because of different and often varying number of dimensions. First, we define quality dimensions for a primitive (i.e., not a composite) data web service:

- ***Response time:*** is the expected delay in seconds between the moment when the request is sent and the moment when results are received.
- ***Availability:*** is the percentage of time that a service is operating.
- ***Reliability:*** is the probability that a request is correctly responded within the maximum expected time frame indicated in the web service description.
- ***Reputation:*** is a measure of its trustworthiness. It mainly depends on end user experiences of using a service s.
- ***Cost execution:*** is the sum of the fee that a service requester has to pay for invoking the service and the fee associated with the usage of the data.

We use the quality criteria defined above to compute the QoS of query plans. Related work has been conducted in [83], where only web services executing specific tasks are considered, so that each component service is executed exactly one time. However, in our context, as we explained earlier, each individual service can be executed as many times as the number of answers resulting from the previous service invocation. To do so, we need to estimate the number of calls to execute for each primitive service in the composition. The number of web calls of a data service s_i in a Plan P_i is the cardinality of the cartesian product of the results returned by different previous services in the plan. We compute it as follows:

$$\begin{aligned} calls\_number(s_i) &= |\prod_j R(s_j)|, s_j \prec s_i \\ |R(s_j)| &= calls\_number(s_1) * avg(s_1) \end{aligned} \tag{4.3}$$

where $s_1, s_2..., s_j$ is the set of services providing required inputs for the execu-

Service	Response Time(sec)	avg
s_1	20	4
s_2	25	5
s_3	20	1
s_4	2	10
s_5	20	4

Table 4.1 – Average Response Time and Results Number of s_1, s_2, s_3, s_4 and s_5

tion of the service s_i, $R(s_1), R(s_2)..., R(s_j)$ represent the corresponding sets of returned results and avg is the average number of results(tuples) returned by each data service s_{kj}.

Consider the following query plan depicted in Figure 4.2 and the corresponding table (Table 4.1) representing the response time and the average number of results that could be returned by each service s_i in a Plan P:

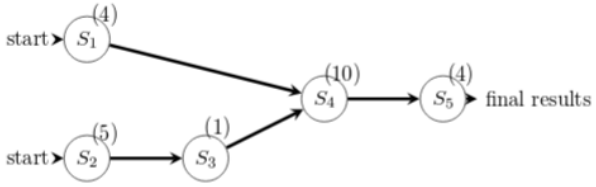


Figure 4.2 – Executable Query Plan

Services s_1 and s_2 must be executed exactly one time, however the service s_3 depends on the results of s_2. By computing the number of calls estimated for s_3 using the formula 5.1, we find that s_2 may output 5 different answers, then s_3 may be invoked 5 times, each time using different input data. s_4 depends on the outputs of both s_1 ans s_3. In such case, we construct all possible combinations between the results of s_1 and s_3, that is a set containing 4 answers returned by s_1 and a set containing 5 tuples returned by s_3, then $calls\_number(s_4) = 20$. The aggregation functions to compute the QoS of a Query Plan in [83] are redefined to include the number of calls (nc) of services in the composition, that is:

- ***Response Time:*** is computed using the Critical Path Algorithm [53]

$$Time(P) = CPA(Time(s_i, op), \ldots, Time(s_n, op)) \tag{4.4}$$

- ***Cost execution:*** is the sum of the total cost execution of each invoked operation over the services that participate in P.

$$Cost(P) = \sum_{i=1}^{n}(calls\_number(s_i, op).Cost(s_i, op)) \tag{4.5}$$

- ***Reputation:*** is the average of all services' reputation in the query plan.

$$reputation(P) = \frac{1}{n}\sum_{i=1}^{n}(reputation(s_i)) \tag{4.6}$$

- ***Reliability:*** is computed using the following formulas:

$$reliability(P) = \prod_{i=1}^{n}(\prod_{k=1}^{nc} \exp^{reliability(s_i)*z_i}) \tag{4.7}$$

- ***Availability:*** is computed as follows:

$$availability(P) = \prod_{i=1}^{n}(\prod_{k=1}^{nc} \exp^{availability(s_i)*z_i}) \tag{4.8}$$

The selection of plans involves multiple criteria (i.e., different QoS criteria). The idea is to associate an aggregated quality score to each executable query plan in order to be able to compare and rank different plans. This problem, from the multi-criteria perspective, can be solved using the Simple Additive Weighting (SAW) [37] method.

SAW technique: is a multi attribute decision technique. The basic logic of the SAW method is to obtain a weighted sum of performance ratings of each alternative over all attributes. The first step consists in computing an evaluation score for each QoS criteria by multiplying the scaled value.

Given a set of executable query plans determined by the Algorithm 9 $P = \{P_1, P_2, \ldots, P_n\}$ and let $C = \{response\_time, reliability, reputation, availability\}$ be the set of QoS criteria; whereas the response time is a negative criterion (i.e., the higher the value is, the lower the quality is), all other criteria are positive (i.e., the higher the value is, the higher the quality is). We decided to not include the execution cost (the financial cost) in the evaluation of the quality of query plans, while it will be later considered in the selection of final plans to execute.

First of all, we associate for each query plan a quality vector representing different criteria values $< duration(P_i), reliability(P_i), reputation(P_i), availability(P_i) >$ Then, we accumulate all quality vectors in a quality matrix Q where each row represents a plan P_i and each column represents a specific QoS criteria.

$$Q = \begin{pmatrix} duration(P_1) & rep(P_1) & rel(P_1) & avail(P_1) \\ duration(P_2) & rep(P_2) & rel(P_2) & avail(P_2) \\ ... & ... & ... & ... \\ duration(P_n) & rep(P_n) & rel(P_n) & avail(P_n) \end{pmatrix}$$

Once the quality matrix is constructed, we proceed to scale criteria values. The value for the response time is computed using the Equation 4.9. For positive criteria, values are scaled according to Equation 4.10.

$$V_{i,j} = \begin{cases} \frac{Q_j^{max} - Q_{i,j}}{Q_j^{max} - Q_j^{min}} \text{ if} Q_j^{max} - Q_j^{min} \neq 0 \\ 1 \text{ if} Q_j^{max} - Q_j^{min} = 0 \end{cases} \tag{4.9}$$

$$V_{i,j} = \begin{cases} \frac{Q_{i,j} - Q_j^{min}}{Q_j^{max} - Q_j^{min}} \text{ if} Q_j^{max} - Q_j^{min} \neq 0 \\ 1 \text{ if} Q_j^{max} - Q_j^{min} = 0 \end{cases} \tag{4.10}$$

where Q^{max} and Q^min represent respectively the maximal and the minimal values of a specific quality criterion in Q.
In the second step, we compute the overall quality score for each query plan using the following formula 5.2:

$$Quality(P_i) = \sum_{j=1}^{k} (V_{i,j} * W_j) \tag{4.11}$$

$W_j \in [0, 1]$ is the weight of each criterion, such as $\sum_{j=1}^{4} W_j = 1$. In our work, we assume that all criteria have the same impact, so we associated the same weight for all criteria (equals to 0.25).

4.3.2 Selection of Query Plans

Our problem can be formulated as follows: Given a set of executable plans, P, each of which is characterized by a quality score and an execution cost $(\mathtt{q_i}, \mathtt{c_i})$, we would like to select a subset `S` of `P` such that the sum of the qualities of the constituent plans is a maximized subject to a given cost threshold, specified by the user. That is:

$$\mathtt{Max} \sum_{\mathtt{i} \in \mathtt{S}} \mathtt{q_i} \ \mathtt{such\ that} \sum_{\mathtt{i} \in \mathtt{S}} \mathtt{c_i} \leq \mathtt{C}$$

Notice that the above problem is a 0-1 knapsack problem, which can be efficiently solved in pseudo-polynomial time $\mathcal{O}(nW)$ using dynamic programming (Algorithm 10). Specifically, let $\mathtt{Q}[\mathtt{i}, \mathtt{w}]$ be an array that stores the maximum combined quality of any subset of plans $\{\mathtt{1}, \mathtt{2}, ..., \mathtt{i}\}$ of a combined cost of at most

c. Our problem can be defined recursively as follows:

- Initial solution

  $\mathtt{Q}[\mathtt{0},\mathtt{c}] = \mathtt{0}$ for $\mathtt{0} \leq \mathtt{c} \leq \mathtt{C}$, i.e., the quality is 0 when no plan is selected,
  $\mathtt{Q}[\mathtt{i},\mathtt{c}] = -\infty$ for $\mathtt{c} < \mathtt{0}$, i.e., a negative cost is illegal

- Recursion step

  $\mathtt{Q}[\mathtt{i},\mathtt{c}] = \mathtt{max}(\mathtt{Q}[\mathtt{i}-\mathtt{1},\mathtt{c}], \mathtt{qi} + \mathtt{Q}[\mathtt{c}-\mathtt{ci}])$ for $\mathtt{0} \leq \mathtt{i} \leq \mathtt{n},\ \mathtt{0} \leq \mathtt{c} \leq \mathtt{C}$

This is the best quality obtained when considering the `ith` plan that can be obtained by not taking this plan, or selecting this plan together with the best solution whose cost is lower than $\mathtt{c} - \mathtt{ci}$.

Algorithme 10 : Select Plans

Require : n: the number of plans, which are labeled 1 to n
q_a: an array containing the values of the qualities of the plans
c_a: an array containing the costs of the plans
C : the maximum cost allowed

Ensure : Q, an array of the best qualities obtained considering a given plan under a cost constraint

```
1  for (c = 0 to C) do
2      Q[0,c] = 0
3  for (i = 1 to n) do
4      for (c = 0 to C) do
5          if ((c_a[i] ≤ c) and (q_a[i] + Q[i-1,c - c_a[i]] > Q[i-1,c])) then
6              Q[i,c] = q_a[i] + Q[i-1,c - c_a[i]]
7              keep[i,c] = 1
8          else
9              Q[i,c] = Q[i-1, c]
10             keep = [i,c] = 0
11 T = C for (i = n downto 1) do
12     if (keep[i,T] = 1) then
13         Print i
14         T = T - c_a[i]
15 Return Q[n,C]
```

Algorithm 10 proceeds as follows. It first initializes to 0 the quality when no plans are selected (line 0). It then iteratively computes the quality under different constraints (from 0 to C) considering different plans (i from 0 to n), see Lines 2

to 13 of the algorithm. Notice that the algorithm also uses a local array named *keep* to identify those plans that were picked up for a given cost c. This is crucial as it allows us later on to identify the plans that were selected and output them (lines 14 to 20). The set of plans finally selected must have together a maximal aggregated quality score w.r.t the maximal cost specified by the user. Thus, the invocation of a larger number of data services may increase the number of answers. However, if the execution of different service compositions may not satisfy the user's requirements, the Algorithm 10 would select only the optimal plan from P.

4.4 Experimentation

We have implemented all the algorithms described in this chapter in Python. In this section, we report on the results of the empirical evaluation that we conducted to assess the effectiveness of our service selection and composition approach.

4.4.1 Testbed and Methodology

The performance of our system is evaluated using publicly available DP services, that export rich information about different domains (e.g., information about books, authors, music, albums, movies and videos, etc.). In particular, we refer to DP services provided by the following websites: LibraryThing[3], ISBNdb[4], AdeBooks[5], IVA[6], LastFM[7], MusicBrainz[8], Discogs[9], MusixMatch[10], Spotify[11], IMDb[12] and Amazon[13],etc. All these websites allow users to query their underlying data based on various criteria such as the QoS, supported by their corresponding service calls. For this, a built-in collection of data service views is defined in the service lake.

4.4.1.1 Profiling Data Services

In order to calibrate the response time of the different data services, we ran a series of service calls. We invoked each service a hundred times using entries selected from the local data source, and then we computed the average response

[3]https://www.librarything.com/
[4]https://isbndb.com/
[5]https://www.abebooks.com/
[6]https://www.internetvideoarchive.com/
[7]https://www.last.fm/api/
[8]https://musicbrainz.org/
[9]https://www.discogs.com/
[10]https://www.musixmatch.com/fr/
[11]https://www.spotify.com/fr/
[12]https://www.imdb.com/
[13]http://aws.amazon.com/publicdatasets/

time for each of them.
In terms of execution cost, since we are using public and free data services, we do not have to pay for invoking the different data services. However, to show how our system selects only query plans not exceeding the user specified cost threshold and maintaining good service quality, we gave different call prices to each service provider. Finally, we used user reviews to compute the reputation of the services and we obtained the reliability and the availability values from either the SLA [67] agreement of data services or their HTML pages.

4.4.1.2 Data Queries

For our experiment, we considered two distinct set of queries: the first contains queries for which none of the requested data nor their answers could be found in the local data source; the second consisted of queries that had a partial answer in the local data source and where the missing data was shuffled. We tested our system using queries on a sundry of topics as input.

Q_1. Find mbid, duration of all the recordings named *'Sorry'*.

Q_2. Find mbid, artist name, url, and ratings of all the albums of name *'Believe'*.

Q_3. Find mbid, artist name, url, streamable, listeners and ratings of all the albums of name *'Believe'*.

Q_4. Find mbid, birth date , nationality, rating of the singer *'Cher'*.

Q_5. Find mbid, birthdate, nationality, address, genres, listeners, tags of rating of the singer *'Cher'*.

Q_6. Find artist mbid and artist url of the album *'Believe'* of the singer *'Cher'*.

Q_7. Find titles and years of movies featuring *'Queen Latifah'*.

Q_8. Find isbn, authors, publisher, editors and topics of all the books entitled *'Principles of data integration'*.

Q_9. Find titles, year of publication, authors and publisher names of all the books dealing with *'data integration'*.

4.4.2 Results and Discussion

To demonstrate the effectiveness and the efficiency of our service selection and composition model, we ran a series of experiments.

4.4.2.1 Enumerated Service Compositions and Query Plans

The first series of experiments aimed at evaluating the two first algorithms, *Algorithm 8* and *Algorithm 9*, for the composition of relevant data services.

	Number of Services	Compositions	Executable Plans	Computation Time (sec)
Q_2	20	10	5	0.06
	40	22	8	0.08
	60	82	36	0.31
	80	293	62	0.57
	100	1209	178	1.73
	200	2293	334	3.25
Q_3	20	43	28	0.38
	40	92	76	0.76
	60	114	114	0.92
	80	372	305	2.56
	100	1216	1070	9.51
	200	2401	1070	9.51

Table 4.2 – Query Planning

For each query, we varied the number of data services from 20 to 100 and measured the number of compositions enumerated in the first stage by *Algorithm 8*, and the number of final candidate plans created using the *Algorithm 9*.
Table 4.2 shows the query planning statistics for queries Q_2 and Q_3, as the number of available data services varies. We note that the number of service compositions generally increases with the total number of data services in the service lake. Indeed, this results from an increasing number of data services relevant to the query. However, the number of executable plans does not always increase with the number of data services available (e.g., considering the query Q_2, the number of executable query plans is the same when there is 100 or 200 data services). This is due to the fact that *Algorithm 9* is effective in pruning away non executable compositions. The total time to generate all possible compositions and create query plans is indicated in Table 4.2. Note that the overall time increases with the number of relevant data services, in other words, with the size of the service lake.

4.4.2.2 Answers & Web calls

The second series of experiments aimed to measure the number of answers, the number of web calls, the time in seconds required to compute the desired answers, and the quality of the selected query plan. We used these criteria as evaluation metrics to validate our results. Furthermore, we compare the results of our knapsack-based selection method (II) to those obtained based on the two classical service selection methods used to answer data queries in the literature: (I) one

executes all possible query plans [66], the other (III) executes only the optimal one [83, 17, 12, 39, 35, 75] (see Chapter 2 for more details). In table 4.3, we show the values obtained for the first four data queries Q_1, Q_2, Q_3 and Q_4, when the cost threshold is fixed to 3$ and 30$ for Q_1 and the rest of the queries, respectively.

		Cost ($)	Time (sec)	Quality	Web calls	Answers
Q_1	Optimal Plan(1)	0.30	0.015	0.68	1	25
	Knapsack M.(4)	2.8	0.108	0.503	9	39
	All Plans(108)	141.7	3.214	0.607	506	43
Q_2	Optimal Plan(1)	0.30	0.015	0.59	1	9
	Knapsack M.(2)	20.4	0.06	0.54	102	15
	All Plans(334)	3268.2	12.198	0.641	13572	19
Q_3	Optimal Plan(1)	10.2	0.032	0.79	51	9
	Knapsack M.(3)	22.8	0.051	0.642	48	19
	All Plans(1070)	9463.1	36	0.487	34729	19
Q_4	Optimal Plan(1)	0.50	0.028	0.586	2	1
	Knapsack M.(8)	2.99	1.207	0.543	21	1
	All Plans(124)	557.5	3.53	0.487	2308	1

Table 4.3 – Results of the Evaluation

The execution of all candidate plans (I) always yields the most answers for all the queries. Unfortunately, the number of web calls, and thus the financial cost, increase considerably, especially for queries in which the composition of different DP services is necessary. The difference is obvious when comparing to the two other methods. For instance, all methods return one answer for Q_4, even though the cost of (I) is much higher than the optimal cost required to compute the desired answers. For Q_3 and Q_4, our service selection method (II) returns the same answers as (I), while for Q_1 and Q_2, it returns less answers. Indeed, for the latter queries, we obtain less information but the computational cost of the additional answers is strictly higher than the execution cost of the other methods (i.e., if we consider Q_3, we need to execute 13470 more web calls to get four more answers). This is because different compositions of service calls retrieve the same data. The number of plans selected by each method are displayed in parenthesis in Table4.3. Using our approach, we try to avoid unnecessary web calls that produce redundant answers.

The execution of the optimal plan (III) often return incomplete answers (e.g., the cases of Q_1, Q_2 and Q_3). Moreover, the different service calls within the optimal

plan may return redundant data for a higher cost. For example, the optimal plan for Q_2 includes two data services of the same provider **LastFm**: *getAlbumInfo* and *getAlbumBysearch*, these two services provide access to the same database, they extract the same triples.

In a nutshell, our approach (II) provides a compromise between (I) and (III). In most cases, it returns more answers than (III), and almost as many answers as (I) for a lower cost.

4.5 Concluding Remarks

In this chapter, we presented a solution to enrich local datasets using data services. In doing so, we used and adapted local-as-view data integration techniques [50, 49]. Moreover, we elaborated a knapsack-based algorithm to select services that yield good quality results without exceeding a given budget (time and monetary cost). The evaluation exercises that we conducted showed the effectiveness of our solution.

Chapter 5

Automatic Specification of Data Services' Views

Contents

User data queries are posed against the local data source that the user is interested in, thereby freeing users from having to interact with the local data source and the data services, each one individually. Nonetheless, the local data source schema and the data services may use different vocabularies to refer the same entity. Therefore, we needed views that relate the capabilities of each data service (I/O parameters) to the concepts, attributes and relations in the local data source. In the previous chapter, we assume that services' views are manually pre-defined in

the Service Lake. This is a time-consuming and tedious task. To address this problem, we propose, in this chapter, a solution to automatically generate views that map the capabilities description of data services into the schema elements of the user data source. In doing so, we only consider the inputs and outputs of data services which are parts of the functional features. To the best of our knowledge, this work is the first to deal with the automation of the definition of service views.

In this chapter, we first give a brief overview of our proposal for the automatic specification of service views in Section 5.1, describing the main components (Section 5.2) , to the developed algorithms (Section 5.3). We then present in Section 5.4 a case study to showcase the working of the proposed algorithm. Finally, we present the experimental evaluation results in Section 5.5.

5.1 Approach Overview

The definition of service views over the user data source is performed in two steps as illustrated in Figure 5.1.

Given a data service description (I/O parameters) and the schema graph corresponding to the user data source, the first step consists of identifying the possible correspondences between the service parameters and the concepts in the schema graph which are semantically related. Such matchings are used as ingredients to simplify the intricate task of views definition. To date, a number of algorithms have been devised and implemented for finding correspondences[14] between schemas. Thus, we rely on automatic matching tools to automatically determine the semantic correspondences between the user data source and the data services. Formally, the match operation determines a mapping indicating which elements of the input schemas logically correspond to each other with a similarity score indicating the plausibility of their correspondence.

Then, based on the matchings obtained, our problem becomes that of finding a tree that covers the service parameters, or more specifically the nodes that match them in the schema graph. Several trees can fulfill such a criterion. Our problem can be, therefore, stated as that of finding the *best* tree. For our purpose, we formulate the search of the *best* tree as that of finding the minimum Steiner tree problem, which we will present later on. To do so, the second step automatically creates a node/edge-weighted graph, depicting the schema of the local

[14]All of the terms correspondences, match candidates and matches refer to matchings.

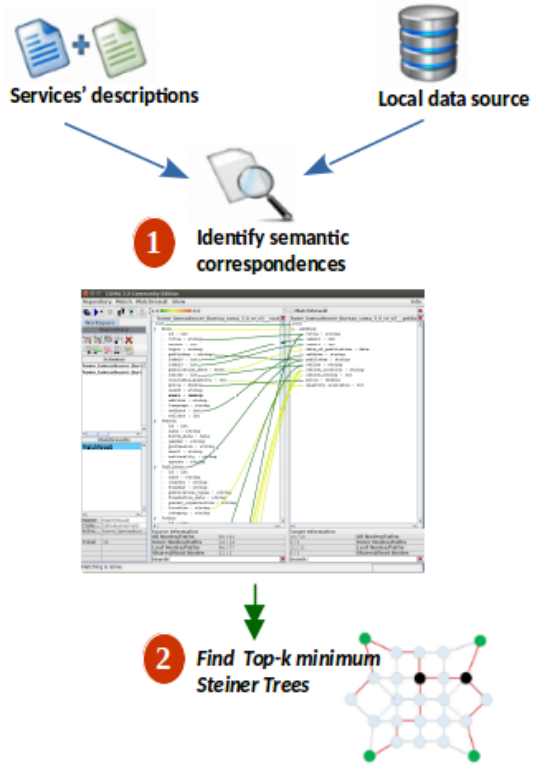


Figure 5.1 – Overview of the Automatic Services' Views Definition Process

data source such as the weights of the nodes represent an aggregated matching score on the node attributes. Then, we study finding the top-k minimum Steiner trees. We provide in the following sections a step-by-step explanation.

5.2 Schema Matching

In the first step, the service schema is compared against the schema of the user data source to identify the semantic correspondences between the underlying database elements and the service parameters. In doing so, we rely on an automatic schema matching tool such as COMA++ [14]. Tools for schema matching take as input the service schema and the schema of the user database and outputs a set of correspondences between their attributes, each of which captures a single pair of matching elements and a similarity value between 0 (strong dissimilarity) and 1 (strong similarity) indicating the plausibility of the match relationship. Within an individual correspondence, one or more service parameters can match one or more concepts from the user database. Thus, we can have multiple match

candidates with different similarities for the same service parameter as shown in Figure 5.2.

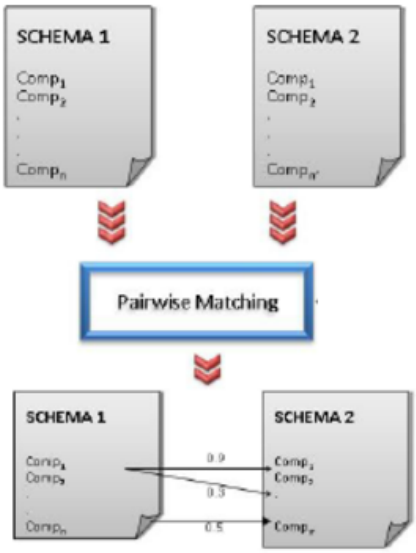


Figure 5.2 – Schema Matching

Example 5.2.1 *if we consider a data service s_1 which given a student Social Security Number (SSN), returns the student name (Name), his level (Level) and his marks (Marks) and a university database DB_u that contains a table Grad-Student(Name, ID, Major, Grades), possible matching would be: $s_1.SSN = DB_u.ID$, since SSN and ID refer semantically to the same real-world entity (the Social Security number of the student). Similarly, $s_1.Name = DB_u.Name$, $s_1.Major = DB_u.Major$ and $s_1.Marks = DB_u.Grades$, all of them represent possible matchings between the schemas of s_1 and DB_u.*

Getting certain semantic mappings between the service parameters and the user data source schema is essential to produce more accurate service views. Unfortunately, automatic schema matching is inherently uncertain because the semantic of schema objects cannot be fully derived from data and meta-data information. Indeed, the correspondences created by automatic schema matching tools are often incomplete and sometimes erroneous. In general it is impossible to identify fully automatically correct mappings and hence post-matching human efforts are always needed to obtain a correct set of correspondences. However, our objective is to automate the whole views definition process. Automatic schema matching tools assign matching scores to candidate correspondences. A matching score may be interpreted as a probability for the correctness of a correspondence. Yet,

matching scores are not normalized, often unreliable and unrelated to the service context. Thus, in this work we use schema matching results as-is such we investigate all the possible candidate matches of service parameters with the aim to explore all the reformulation possibilities.
Formally, a service operation $(\mathtt{op}, \mathtt{L_{in}}, \mathtt{L_{out}})$ is characterized by a name, $\mathtt{op}$, and a set of input and output parameters, $\mathtt{L_{in}}$ and $\mathtt{L_{out}}$. In what follows we use $\mathtt{l}$ to denote the magnitude of the union of $\mathtt{L_{in}}$ and $\mathtt{L_{out}}$. Note that a node in the schema graph is a tuple with several attributes, a node contains a service parameter if one of its candidate matchings appears in any of the attributes of the corresponding tuple, and a node may contain several service parameters. Thereafter, matching a service operation $(\mathtt{op}, \mathtt{L_{in}}, \mathtt{L_{out}})$ with a schema graph yields the following weighted sets:

- $\mathtt{V_1}, \ldots, \mathtt{V_l}$ represent the sets of vertices in $\mathtt{V_s}$ that match the parameters in $\mathtt{L_{in}}$ and $\mathtt{L_{out}}$. More specifically, $\mathtt{V_i} \subseteq \mathtt{V}$ is a set of vertices containing the attributes matching a given parameter p_i, for $\mathtt{i} = 1, \ldots, \mathtt{l}$
  Note that a vertex represents a concept with several attributes, it may contain multiple attributes that correspond to different service parameters.
- $\mathtt{V_{op}}$ represents the sets of vertices in $\mathtt{V_s}$ that match the operation $\mathtt{op}$

The elements of the sets $\mathtt{V_1}, \ldots, \mathtt{V_l}$ and $\mathtt{V_{op}}$ are labeled with a matching score. Given these matching scores, the edges in the graph $\mathtt{G_s}$, $\mathtt{E_s}$, are also labeled with a score.

5.3 Automatic Specification of Services' Views

The objective of this step is to determine the most accurate reformulation of services' descriptions on top of the user data source schema by means of Steiner trees. In the following, we start by giving some brief background on the Steiner Tree Problem.

5.3.1 Steiner Trees Problem in Graphs

The Steiner Tree Problem (STP) is an important problem in combinatorial optimization which has numerous applications, ranging from the design of integrated circuits, evolution theory to the computer networking, etc. A Steiner tree is an acyclic connected subgraph that spans a given subset of vertices. This problem can be formulated in terms of an undirected graph G=(V,E) where V is the set of

vertices, E is the set of edges, and S $\subseteq$ V is a subset of vertices called terminals. Given a cost function c: E $\mapsto \mathbb{R}$, the minimal Steiner tree problem asks for a tree, T=(V', E'), with V' $\subseteq$ V and E' $\subseteq$ E, spanning all the vertices of S such that $\sum_{e \in E} c(e)$ is minimum. Note that a Steiner tree may include some vertices of V\S. These are known as Steiner vertices.

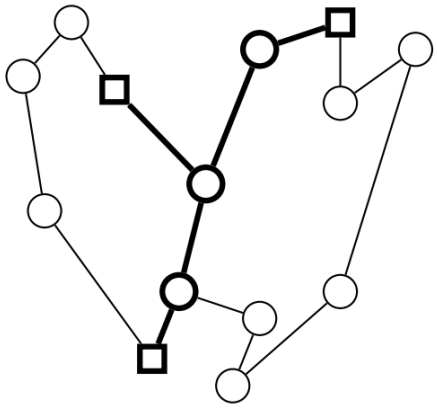


Figure 5.3 – An Example of a Steiner Tree Problem

An example of a Steiner tree problem is shown in Figure 5.3, where the terminal vertices are denoted by rectangles. The highlighted vertices and edges represent a Steiner tree for the given problem instance, covering all the terminals and including three Steiner nodes.

Many different variants and generalizations of the Steiner tree problem have been investigated in the literature. In this work, we are particularly interested in the generalized version of STP in graphs, the so-called the Group Steiner Tree Problem (GSTP).

5.3.1.1 Group Steiner Problem in Graphs

Motivated by the wire routing phase in physical VLSI design, the GSTP was introduced by Reich and Widmayer [70] as a generalization of the STP, such given an undirected graph and several subsets of required vertices called groups, the goal is to find a shorted connected subgraph connecting at least one required vertex from each group. The groups need not to be mutually disjoint vertex sets. An example is shown in Figure 5.4.
The example includes a graph with three groups $\mathtt{Q} = \{\mathtt{Q}_1, \mathtt{Q}_2, \mathtt{Q}_3\}$; the vertices within each dotted region represents a group. The highlighted tree represents a group Steiner tree connecting at least one vertex from each group, which is the

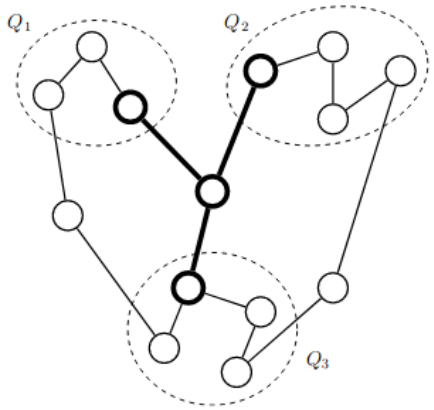


Figure 5.4 – A Group Steiner Tree Connecting at Least One Vertex from Each Group Q_1 ,Q_2 and Q_3

only possible solution for this problem instance. Note that it is possible to have more than one group Steiner tree for a different problem instance.

The GSTP can be stated formally as follows: given a graph $\mathtt{G} = (\mathtt{V}, \mathtt{E})$ with the cost function c: E $\mapsto \mathbb{R}$ and groups of vertices $\mathtt{g} = \{\mathtt{g_1}, \mathtt{g_2}, \ldots, \mathtt{g_k}\}$, the objective is to find the tree T of G with the minimum cost that contains at least one vertex from each of the sets $\mathtt{g_i} \in \mathtt{g}$. Formally, find T = (V',E') that minimizes $\sum_{e \in E'} c(e)$ such that for all i $\in$ {1,. . . ,k}.

5.3.2 Automatic Definition of Services' Views

Given a schema graph $G_s = (V_s, E_s)$ and the matching results computed using a matching tool, we define a weighted graph G(V,E), where V and E are the set of vertices and edges in G_S, respectively. In doing so, we weight nodes and edges as follows (Eq.(5.1), respectively Eq.(5.2)):

$$w_v(v) = 1 - \frac{\sum_{p_i \in v} match(p_i)}{\sum(p_i \in v)} \tag{5.1}$$

Where $\frac{\sum_{p_i \in v} match(p_i)}{\sum(p_i \in v)}$ is the average matching score of all attributes $\in v$ matching different service parameters. Then $w_v(v)$ represents the non matching score and therefore our aim is to find trees with nodes having the minimal weight as possible.

$$w_e(u, v) = 1 - \frac{\sum_{p \in p_2 - p_1} match(p)}{Card(p_2 - p_1)} \tag{5.2}$$

where p_1 and p_2 are respectively the set of parameters having matching attributes in u and v. $p_2 - p_1$ is the difference between p_1 and p_2, in other terms, the set of parameters matching provided by the node v but that do not have any possible matching in u.
Thereafter, generating services' views given the matchings obtained amount mainly to identifying the Steiner trees entailed by such matchings in the graph $\mathtt{G_s}$. A Steiner tree can be viewed as the smallest tree connecting a given vertices in a graph. The relevance of a solution tree T can be measured using the equation Eq.5.3. Below, we use V(G) and E(G) to denote the set of nodes and the set edges of the graph G, respectively.

$$\mathtt{Rel(T)} = \sum_{\mathtt{v} \in \mathtt{V(T)}} \mathtt{score(v)} \tag{5.3}$$

Enumerating all entailed Steiner trees as well as finding the best Steiner tree are NP-Complete problems, i.e., there is no efficient way to identify them. This has been proven by reducing the problem to a minimum set cover problem [69]. There is therefore a need for a heuristic or approximation algorithm. In particular, in our case we will be interested in using a method that allows us to identify the top-k Steiner trees, given that enumerating all of them is not possible, and more importantly, not useful in our case. We formally state our problem in the following.

Problem Reformulation: Given l service parameters, find the top-k minimum cost connected Steiner trees $\mathtt{T_1}, \ldots, \mathtt{T_k}$ ranked with their relevance $\mathtt{Rel(T_k)} \leq \mathtt{Rel(T_{k-1})} \leq \ldots \leq \mathtt{Rel(T_1)}$. Such that:

7.1) $\mathtt{V_i} \cap \mathtt{V(T_j)} \neq \emptyset$, for $\mathtt{i} = 1, \ldots, \mathtt{l}$ and $\mathtt{j} = 1, \ldots, \mathtt{k}$, and

7.2) $\mathtt{V_{op}} \cap \mathtt{V(T_j)} \neq \emptyset$ for $\mathtt{j} = 1, \ldots, \mathtt{k}$

The first condition ensures that a solution tree cover concepts that represent all the inputs and outputs of the service operation. Whereas the second condition ensures that there is at least one node in the tree that has some correspondence with the operation.

Consider the schema graph depicted in Figure 5.5 in which we represent only concepts, each one is labeled with the set of possible parameters matchings. For simplicity, we represent an attribute matching a parameter p_i by the parameter name p_i, for i=1,...,l. We denote match(p_i) the matching score of a given attribute att $\in v.attributes$ with the parameter p_i.

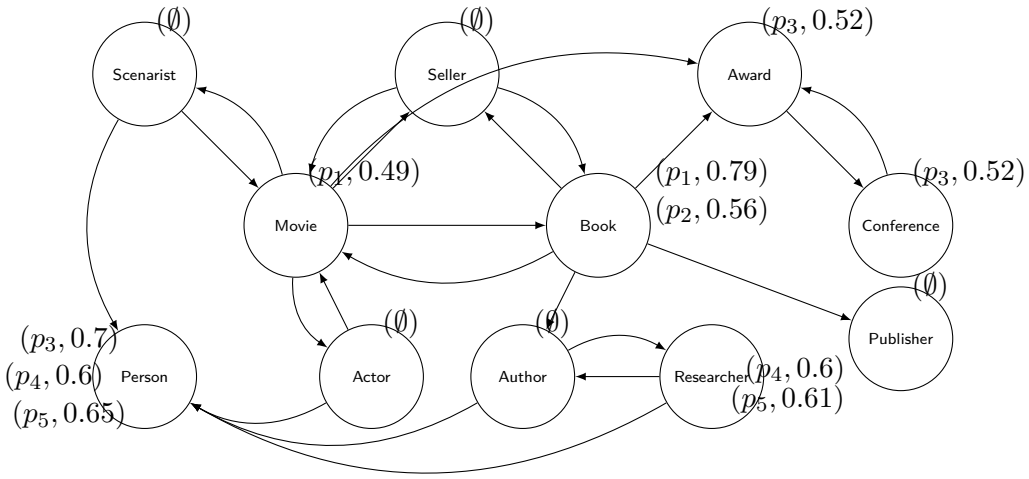


Figure 5.5 – Overview of the Local Database Schema: Nodes and Edges

Our main aim is to find the top-k Steiner trees interconnecting different v_i and v_{op}. In fact, the weight associated to each node reflects the score by which a node is too far to match the data service parameters: the smaller the score, the higher the probability that the corresponding node V_i is relevant for the service view description.

We implement a dynamic programming solution by taking the heights, h, of trees as stages, and find the top-k Steiner trees by expanding the trees with height h=0,1,2, ...,H until k optimal trees are found. H is the largest number of nodes which can be traversed to go from a given V_{op} to any node V_i such as $\mathtt{H} = \max_{V_{op},V_i} d(V_{op}, V_i)$, where $d(V_{op}, V_i)$ is the distance/ shortest path between V_{op} and V_i.

Given a schema graph G_S, a set of service parameters' matchings V_i and a set of service operation matchings V_{op}, Algorithm 11 outputs the top-k minimum cost connected trees that contain all the nodes in V_{op} and at least one node in every subset V_i. The optimal Steiner tree for mapping the data service description to the user data source schema is found from the top-k Steiner trees computed by the algorithm, for subsets of service parameters $p_i \subseteq$ p, with heights $\leq$ H.

Let T(v,p,h) be a tree rooted at node v with height h, containing a non-empty set of parameters p. Every single node v in G containing a nonempty set of parameters, p ($\subseteq$ P), is considered as a rooted tree with zero height, h = 0. Such a tree does not have any edges, and therefore the cost of the tree is the

Algorithme 11 : Top-k Service Views

Require : Schema graph $G_S = (V, E)$, the set of service operation matching V_{op}, the set of service parameters matching $V_1, \ldots, V_l$

Ensure : Q_k : a priority queue containing the Top-K Steiner trees sorted in the decreasing order of matching scores

```
1  Let Q_T be a priority queue sorted in the increasing order of costs
2  Q_T, Q_k ← ∅
3  h= 0
4  H = max_{V_i,V_op}(d(V_i, V_op))
5  for each v ∈ (V ∩ V_i) do
6      enqueue T(v,p,0) into Q_T
7  while (Q_T ≠ ∅) and (h < H) do
8      dequeue Q_T to T(v,p,h)
9      if (Card(p) = l) and (V_op ⊆ V(T)) then
10         enqueue T(v,p,h) into Q_k
11     h ← h+1
12     if V_op ⊈ V(T) then
13         compute T_g(v, p, h) from its neighbors (Eq.(5.6))
14         update Q_T with the new T_g(v, p, h)
15     compute T_m(v_op, p ∪ p', h) (Eq.(5.7))
16     update Q_T with the new T_m(v, p ∪ p', h)
```

corresponding weight of the node v, as given below Eq.(5.4).

$$T(v, p, 0) = w_v(v) \tag{5.4}$$

Here, the left side is the tree, and the right side is its cost. Otherwise, if the node does not contain any parameter p, its cost is T(v, $\emptyset$) = 1. In general, the (minimum) cost of a tree T(v, p,h) is given below in Eq 5.5,

$$s(T) = 1 - \frac{(1 - T(v_0, p_0, 0)) * Card(p_0) + \sum_{e_i \in E(T)} (1 - w_e(e_i)) * Card(p_i)}{Card(p)} \tag{5.5}$$

Where p_i is the set of additional parameters matchings to be brought while considering the (relations/) edge e_i.

Remark: s(T) is a number between 0 and 1 and 1-s(T) represent the matching score of T to the data service.

Algorithm 11 maintains trees in a priority queue Q_T, by the increasing order

of costs. The smallest cost tree is maintained at the top of the queue Q_T. We manipulate the queue using three operators: Enqueue, Dequeue, and Update.

- Enqueue inserts a tree T(v, p,h) into the queue Q_T and Q_T is updated to maintain the increasing order of costs of trees.
- Dequeue removes the top tree T(v, p,h) in Q_T.
- and finally Update first enqueues T(v, p,h) if it does not exist in Q_T and updates Q_T to maintain the increasing order of costs.

Initially Q_T is empty, Algorithm 11 first enqueues all rooted trees T(v, p,h) into Q_T such as v contains a subset of parameters p ($\subseteq$ P) (Algorithm 2, lines 5-6). While the queue Q_T is non-empty and the trees height does not reach H, the algorithm repeats to dequeue/enqueue trees in the attempt to make all trees grow/merge individually to reach a tree containing all the service parameters (their matchings). It iteratively dequeues the top tree T(v, p) from Q_T, which has always the smallest cost among all trees rooted at v and containing the same set of parameters p in Q_T.

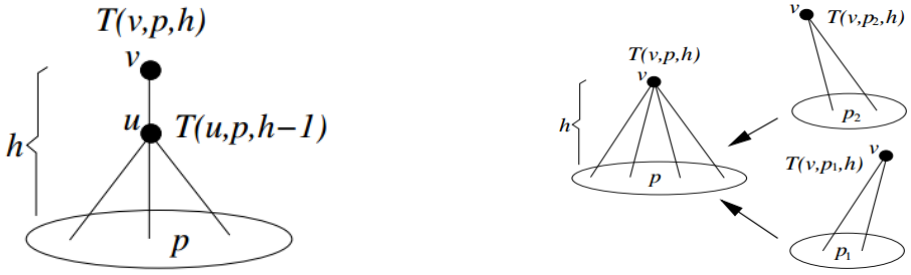
Tree Grow: considering all the neighbors v $\in$ N(u) of a node u, a neighbor may or may not contain some additional parameters matchings p'. If yes, then the left side of the equation 5.6 should be $T_g(v, p \cup p', h)$. Note that T(v, p) may or may not exist in Q_T. If not, T(v, p) = T(u, p)$\oplus$ (u, v) is enqueued into Q_T. Then Q_T will be updated. In both cases, Q_T ensures the increasing order of costs. The case when u does contain some parameters p_0 is similar. The line 14 handles the case of tree grow (Eq. 5.6).

$$T_g(v, p, h) = \min\{(v, u) \oplus T(u, p, h-1) | u \in N(v)\} \tag{5.6}$$

Tree Merge: The case of tree merge (Eq. 5.7) is handled in line 17. As we are interested to find Steiner trees having V_i as leaf nodes, our manner to do, is to search for trees rooted at a given V_{op}. There are trees rooted at the same node v_{op}, each one containing different subsets of parameters: p_1, $p2$, the algorithm considers every possible disjoint pair of p_1 and p_2, T(v, $p_1 \cup p_2$). We use $\oplus$ to denote the operation to merge two trees into a new tree, and N(v) is the set of neighbors of v in the schema graph G_S such as N(v)=$\{u|(v, u) \in E(G_S)\}$.

$$T_m(v, p_1 \cup p_2, h) = \min\{T(v, p_1, h) \oplus T(v, p_2, h) | p_1 \cap p_2 = \emptyset\} \tag{5.7}$$

Then Q_T will be updated. In both cases, Q_T ensures the increasing order of



(a) Tree Grow (Tg) (b) Tree Merge (Tm)

Figure 5.6 – Basic Operations used for the Construction of Steiner Trees

costs. If a given T(v,p) contains all V_i and V_{op}, the algorithm enqueues it in the queue Q_k containing top k optimal Steiner trees such it keeps always the Steiner tree with the higher matching score in the top of the queue. It is important to mention that if the number of found Steiner trees exceeds the size of Q_k, trees with the lower matching score are removed from Q_k and new trees with higher score are added to Q_k.

In fact, the matching score of a tree T(v,p,h) is simply deduced from its score as follows (Eq.(5.8)):

$$match(T) = 1 - s(T) \tag{5.8}$$

We analyze the time and space complexities of *'Top-Service Views'* Algorithm in the following paragraphs.

Time Complexity Let T(v,p) be the minimum cost for a tree rooted at every v ∈ V(G) containing a subset of service parameters, p⊆P where l = |P|. There are totally n nodes, m edges and 2^{l*R} subsets of P where R = $\max_{p_i \in \mathrm{P}}(|matchs(p_i)|)$. The length of Q_T will be at most 2^{l*R}. Note that all trees can be enqueued/dequeued into/from Q_T at most once. Eq.(5.6) is computed in lines 12-14 to grow T(v,p). The total number of possible neighbors to be considered is bounded by O(|N(v)|) such |N(v)| is the number of neighbors of v (line 13). Q_T need to be updated in O(1) time and therefore the total time needed for tree_grow is O(m). Eq.(5.7) is computed in lines 15-16 to merge two trees T(v,p_1) and T(v,p_2). For every pair of non-empty disjoint set p_1 and p_2. Let consider T(v,p_1), the total number of possible trees T(v,p_2) to merge is bounded by O(l-|p_1|). Thus, the total time needed for tree_merge is O(2^l-(l+1)) = O(2^l). So the worst-case complexity is of O($2^{l*R} * (2^l + m)$) = O(2^{l^2*R}); under the assumption that deleting, inserting and updating a Steiner tree in Q_T can be realized in O(1).

Space Complexity T(v,p) represents a subtree in G_S. Thus, we do not need to store the whole tree in memory. We only record the edges from which T is constructed and the set of parameters' matchings it covers. Therefore, the space needed for storing T(v,p) is bounded by O(1) and hence the maximum size of Q_T is bounded by O(2^{l*R}), resulting in a space complexity of O(2^{l*R}) in the worst case. Consider a data service with four I/O parameters (l=4) and given ground truth matchings, we have exactly one matching per each parameter in the local database (R=1). The top-k Steiner trees are obtained from the 8 constructed trees in Q_T .Now, suppose that each parameter has at max three candidate correspondences (R=3). In the worst case, $2^{(3*4)}$ trees are constructed and enqueued in Q_T, and from which the best trees are selected.

Given a large number of matching candidates, our algorithm consumes much more memory and time, as it constructs all possible Steiner trees covering service parameters. Therefore, the less matching candidates we have as input, the better our algorithm performs. Nonetheless, time and space complexities do not reflect the quality of the algorithm outputs (top constructed Steiner trees and hence the quality of corresponding reformulations). Our aim is to output the best Service reformulations (top Steiner trees) in priority and not optimizing the algorithm performance.

Top-k Steiner tree computation has been previously studied in the context of keyword search on relational databases [22, 46, 51, 30]. Given an l keyword query, p_1, p_2, ..., p_l against a relational database modeled as a weighted graph, the proposed algorithms study finding top-k minimum cost connected trees that contains all keywords. For our purpose, we adopted the same strategy as given in [30]. The proposed algorithm DPH-1 explores all Steiner trees with heights h $\leq$ H = diameter(G) = O(n), from which the optimal GST-1 (Group Steiner Tree) is found. The main problem of DPH-1 is that it cannot determine whether the current minimal cost Steiner tree spanning all the keywords is the optimal until h reaches its maximal value, H. For this purpose, the authors proposed another solution with a best first strategy (DPBF-1) to reduce the space complexity required by the DPH-1 algorithm. With the best-first strategy, the DPBF-1 algorithm can terminate once it finds a connected tree containing all the searched keywords such that the first tree found is the optimal Steiner tree. However, this is not useful in our case such we can not terminate the algorithm once a connected tree connecting all service parameters is found. Actually, the candidate semantic correspondences for a service parameter p_i can be scattered in different nodes in the graph. Thereafter, the first tree found would be the minimal cost tree spanning all service parameters through a little number of nodes (with a

height h'), whereas there may exist another connected tree with a less cost but including more nodes to get the appropriate matching from the graph (having a height h $\leq$ h'). Moreover, we cannot ensure that the optimal Steiner tree is always the tree with the minimal score (with the highest matching score) given the uncertainty about the matching scores computed by matching tools. Thus, we choose to select the top-k Steiner trees from the found ones, such k = max(|matches(p_i)|) taking into consideration the minimal number of different reformulations, a service parameter can participate in. k can also be specified by the user (the database expert).

5.4 Case Study

Continuing with the schema graph depicted in Figure 5.5, and consider the service operation *GetBook* from *AbeBooks.com*[15], which given a book title, returns its isbn, the authors names, the publisher name, the publication date, the available quantity, the seller name, its location and rating. We denote the set of input and output parameters as follows: $p = \{p_1(title), p_2(isbn10), p_3(isbn13), p_4(authors), p_5(publisher), p_6(date\ of\ publication), p_7(quantity\ available), p_8(seller), p_9(seller\ location), p_{10}(seller\ rating),\ p_{11}(\text{price})\}$.
After analyzing the operation name and the corresponding textual description, we determine the set of service matching, $V_{op} = \{Book\}$.

The following step is to identify semantic correspondences between the local database schema and the service parameters. Schema matching can be done manually by an expert or (semi-)automatically using a matching tool. To show the effectiveness of our algorithm system, we show in a first series of experiments the top service views generated based on a ground truth (GT) matches we manually specified. In a second series of experiments, we rely on matching results computed using COMA++ [14], one of the best schema matching tools [31]. We demonstrate in the subsequent sections the working of Algorithm 11 using different matching results.

5.4.1 Generation of Service Views using Ground Truth Matching

Despite the big number of research and commercial matching approaches automating most of the matching process, computed correspondences typically need to be validated and corrected by users to achieve the correct match mappings.

[15]https://www.abebooks.com/

Thus, we rely in a first try on a domain expert to match the services' parameters to the target database schema, in order to provide a basis for evaluating the quality and the effectiveness of our algorithm proposed for the generation of services' views. Table 5.1 shows the correspondences between the parameters of the service *getBook* and the local database schema (see Figure 5.5), specified manually by the expert. Each service parameter has exactly one correspondence from the local database with a matching score equals to 1.

matching: service $\longleftrightarrow$ *localdatabase*	score
getBook.title $\longleftrightarrow$ *Book.title*	1
getBook.isbn10 $\longleftrightarrow$ *Book.isbn*10	1
getBook.isbn13 $\longleftrightarrow$ *Book.isbn*13	1
getBook.date of publication $\longleftrightarrow$ *Book.publication_date*	1
getBook.price $\longleftrightarrow$ *Book.price*	1
getBook.quantity_available $\longleftrightarrow$ *Book.available_quantity*	1
getBook.authors $\longleftrightarrow$ *Person.name*	1
getBook.publisher $\longleftrightarrow$ *Publisher.name*	1
getBook.seller $\longleftrightarrow$ *Seller.name*	1
getBook.seller_location $\longleftrightarrow$ *Seller.country*	1
getBook.seller_rating $\longleftrightarrow$ *Seller.rating*	1

Table 5.1 – Schema Matching Defined Manually by an Expert

We first start by demonstrating the 'Top-k Service Views' algorithm. The corresponding groups V_i are the following:

- $V_{title} = V_1 = \{Book\}$
- $V_{isbn10} = V_2 = \{Book\}$
- $V_{isbn13} = V_3 = \{Book\}$
- $V_{dateofpublication} = V_4 = \{Book\}$
- $V_{quantityavailable} = V_5 = \{Book\}$
- $V_{authors} = V_6 = \{Person\}$
- $V_{publisher} = V_7 = \{Publisher\}$
- $V_{seller} = V_8 = \{Seller\}$
- $V_{sellerlocation} = V_9 = \{Seller\}$
- $V_{sellerrating} = V_{10} = \{Seller\}$
- $V_{price} = V_{11} = \{Book\}$

Algorithm 11 starts by computing H, the maximal number of nodes a Steiner tree can contain. In the algorithm, H represents the maximal number of tree-grow that can be applied on a given tree (a rooted tree). In this case, H=2, which

means that the algorithm will stop processing once h becomes equal to 2. In the following, we will illustrate how our algorithm works in three main iterations (when h=0, h=1 and finally when h=2).

As shown in Figure 5.7, the algorithm 11 initially enqueues four rooted trees in Q_T (lines 5-6); T(v_{Book}, {p_1, p_2, p_3, p_6, p_7, p_{11}}, 0), T(v_{Person}, {p_4}, 0), T($v_{Publisher}$, {p_5}, 0), T(v_{Seller}, {p_8, p_9, p_{10}}, 0). We recall that the priority queue keeps always the tree with the smallest cost S(T), in other words with the higher matching score, in the top.

All these rooted trees are dequeued and new trees are constructed and enqueued

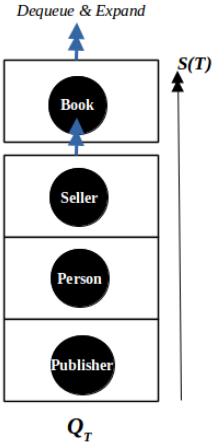


Figure 5.7 – Construction of Rooted Trees

into Q_T based on the case of tree-grow (see Figure 5.8). The algorithm keeps expanding the trees with heights h=1,2. Thereafter, it proceeds to merge the constructed trees with the aim of finding the top-k minimum cost Steiner trees covering all the parameters of the service *getBook*. Figure 5.9 shows some of the constructed trees after the merge operation.

Two Steiner trees were found with a matching score equals to 1. Note that the first Steiner tree, depicted in Figure 5.11b, is the same as the view we manually defined in Chapter 4. The second tree includes one more node *'Movie'*, a Steiner node used as a bridge to reach the node *'Seller'*. Such node can be removed without affecting the quality of the resulting reformulation, that is the Steiner tree 1.

Finally, two service views were generated for the service *getBook* with a matching score equals to 1 (see Listing 5.1 and Listing 5.2).

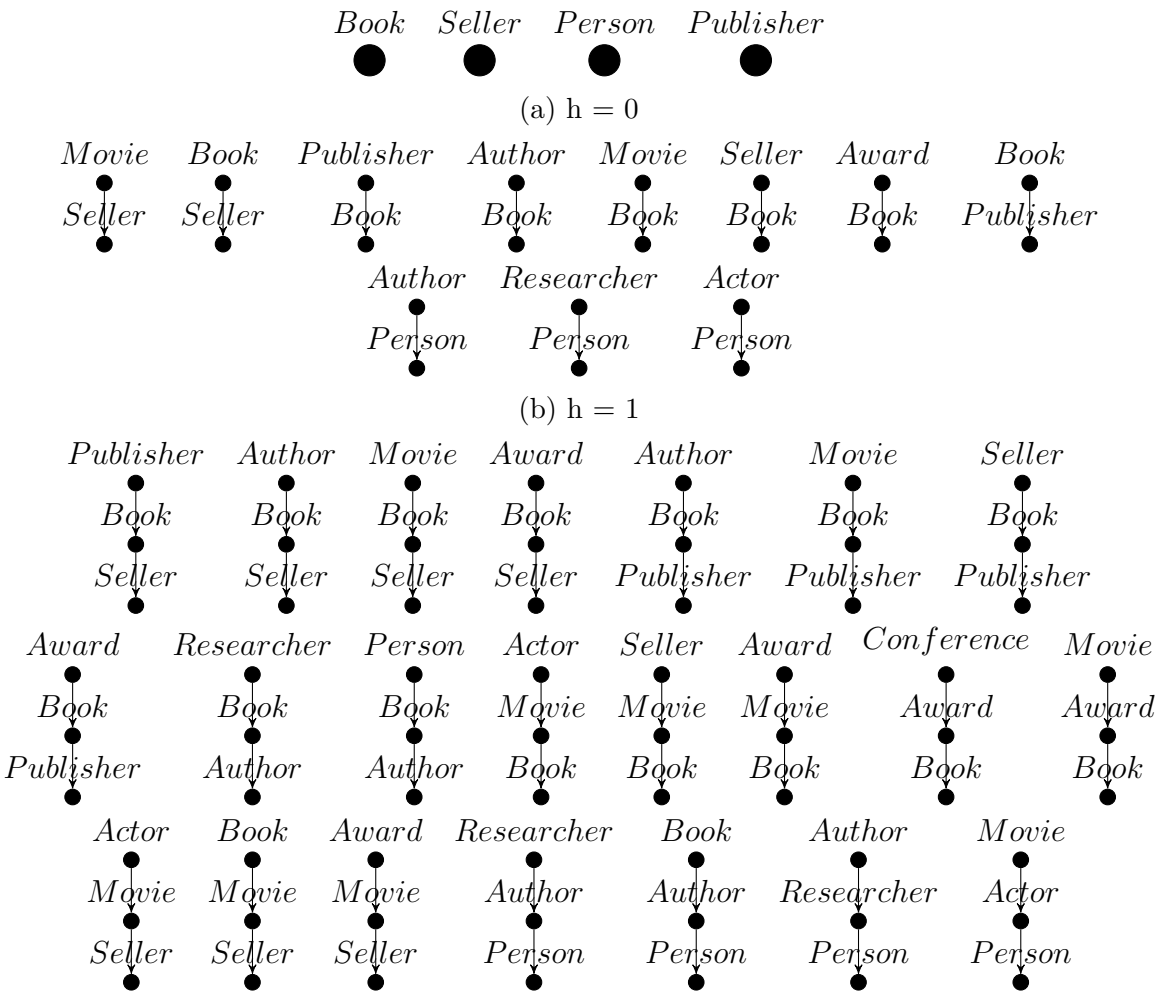
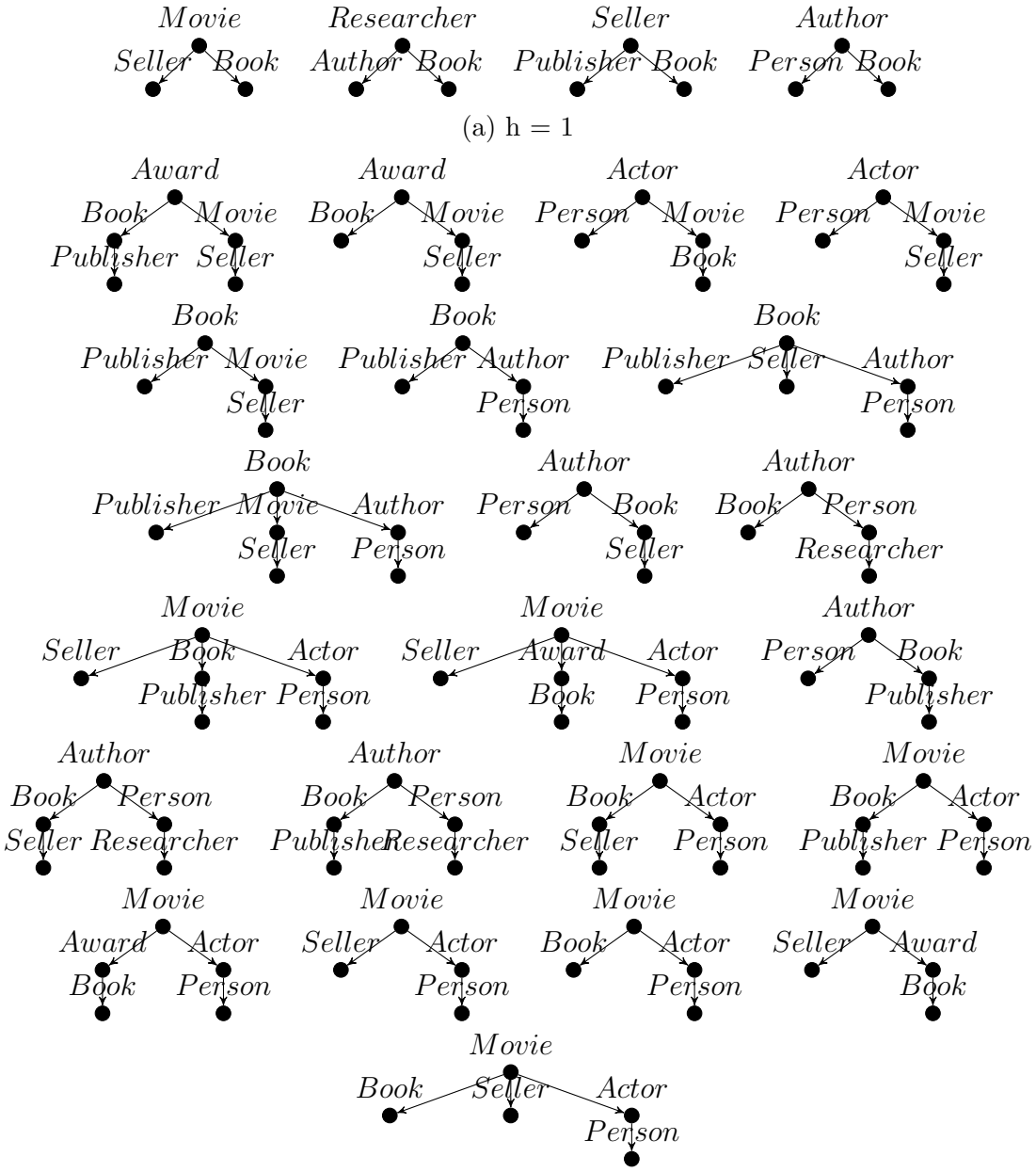


Figure 5.8 – Tree Grow

```
1 VgetBook:Seller(o1),rating(o1,p1),name(o1,p2),location(o1,p3),Publisher(o2),name
    (o2,p4),Book(o3),isbn10(o3,p5),isbn13(o3,p6),title(o3,p7),price(o3,p8),
    publication_date(o3,p9),available_quantity(o3,p10),Person(o4),name(o4,p11),
    Author(o5),have(o1,o3),id(o1,p12),seller(o3,p13),published(o2,o3),id(o2,p14)
    ,publisher(o3,p15),works_as(o4,o5),id(o4,p16),id(o5,p17),wrote(o5,o3),author
    (o3,p18)
```

Listing 5.1 – User-centric View of the service *getBook*, representing the Steiner tree 1

```
1 VgetBook:Publisher(o1),name(o1,p1),Author(o2),Movie(o3),Seller(o4),name(o4,p2),
    rating(o4,p3),location(o4,p4),Person(o5),name(o5,p5),Book(o6),isbn10(o6,p6),
    isbn13(o6,p7),title(o6,p8),price(o6,p9),publication_date(o6,p10),
    available_quantity(o6,p11),published(o1,o6),id(o1,p12),publisher(o6,p13),
    wrote(o2,o6),id(o2,p14),author(o6,p15),based_on(o3,o6),id(o3,p16),movie(o6,
    p17),have(o4,o3),movie(o4,p18),works_as(o5,o2),id(o5,p18)
```



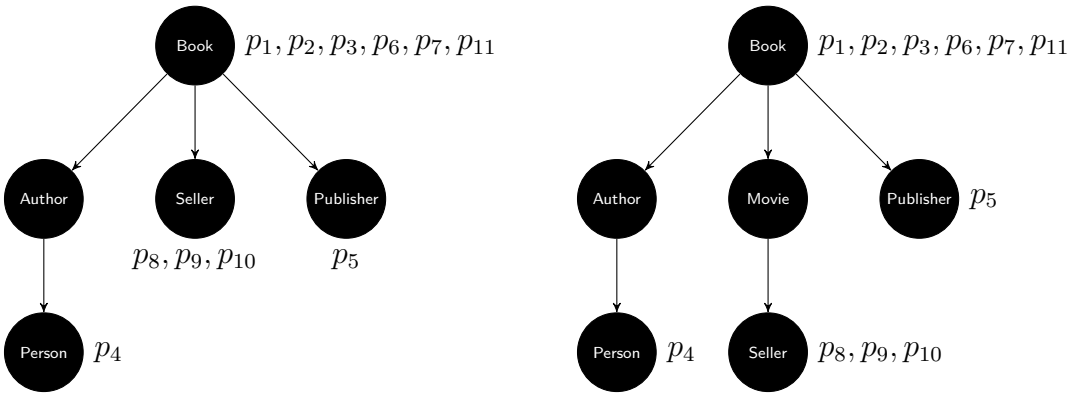
(a) h = 1

(b) h = 2

Figure 5.9 – Tree Merge

Listing 5.2 – User-centric View of the service *getBook*, representing the Steiner tree 2

We notice the existence of additional conjuncts such as works_as(o4,o5), id(o4,p14) and wrote(o5,p16) in Listing 5.1. These conjuncts represent the relationships be-



(a) Top 1 with a matching score = 1.0 (b) Top 2 with a matching score = 1.0

Figure 5.10 – Generated Steiner Trees

tween the different nodes (e.g., foreign keys in relational databases, edges in graph databases, etc.).

5.4.2 Generation of Service Views using Approximate Matching

In the past, schema matching is largely performed manually by domain experts, and therefore a time-consuming and tedious process. To reduce the manual effort required, many techniques and prototypes have been developed to (semi-)automatically solve the match problem. The proposed approaches typically exploit metadata (e.g. schema characteristics such as element names, data types and structural properties), characteristics of data instances, as well as background knowledge from dictionaries and thesauri. In the following, we illustrate the views generation step given the matching results of one selected COMA++ matcher.

5.4.2.1 Schema Matching using COMA++

COMA++ is a schema and ontology matching tool that supports different match strategies. In the following, we apply the *NodesNames* matcher to compute and identify the possible semantic correspondences of each service parameter in the local database.

NodesNames Matcher: This matcher only considers the element names. It first performs some pre-processing steps such as the tokenization to derive a set of components (tokens) of a name and it expands abbreviations and acronyms. The Name matcher then applies multiple simple matchers, such as Affix, Trigram, and

Synonym, on the token sets of the names and combines the obtained similarity values for tokens to derive similarity values between element names [32]. Table 5.2 summarizes the matching results computed using the NodesNames workflow of Coma++.

matching: service ⟷ local database	score
getBook.title ⟷ Book.title	1.0
getBook.title ⟷ Movie.title	1.0
getBook.isbn10 ⟷Book.isbn10	1.0
getBook.isbn13 ⟷ Book.isbn13	1.0
getBook.authors ⟷ Book.author	0.7
getBook.authors ⟷ Author.id	0.7
getBook.publisher ⟷ Book.publisher	1.0
getBook.publisher ⟷ Publisher.id	1.0
getBook.date_of_publication ⟷ Book.publication_date	0.56
getBook.quantity_available ⟷ Book.available_quantity	0.65
getBook.price ⟷ Book.price	1.0
getBook.seller ⟷ Book.seller	1.0
getBook.seller ⟷ Seller.id	1.0
getBook.seller_location ⟷ *Seller.location*	1.0
getBook.seller_rating ⟷ *Seller.rating*	0.52
getBook.seller_rating ⟷ *Movie.rating*	0.52

Table 5.2 – NodesNamesW Matching Results

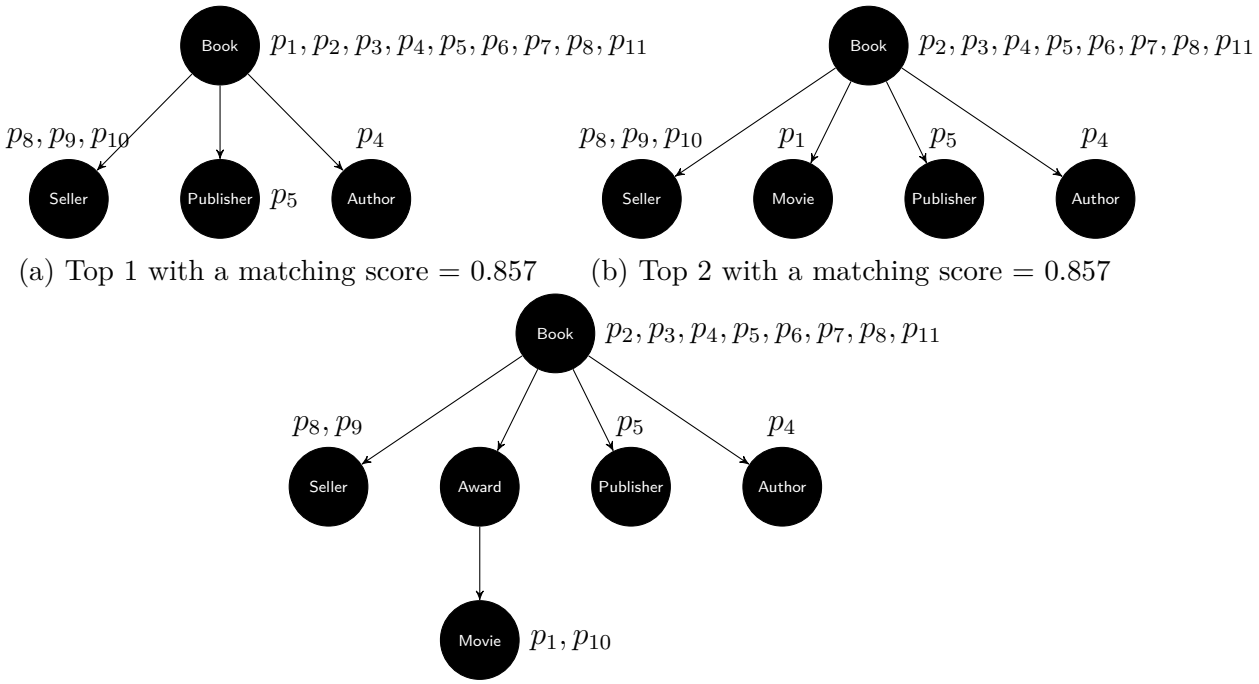
The generated service views are depicted in the followings listings: Listing 5.3, Listing 5.4 and Listing 5.5.

```
1 VgetBook: Seller(o1),rating(o1,p1),location(o1,p2),id(o1,p3),Publisher(o2),Book(
      o3),isbn10(o3,p4),isbn13(o3,p5),title(o3,p6),price(o3,p7),author(o3,p8),
      seller(o3,p9),publication_date(o3,p10),available_quantity(o3,p11),Author(o4)
      ,have(o1,o3),published(o2,o3),id(o2,p12),publisher(o3,p13),wrote(o4,o3),id(
      o4,p14)
```

Listing 5.3 – User-centric View of the service *getBook*, representing the Steiner tree 1

```
1 VgetBook: Seller(o1),rating(o1,p1),location(o1,p2),id(o1,p3),Movie(o2),title(o2,
      p4),Book(o3),isbn10(o3,p5),isbn13(o3,p6),price(o3,p7),author(o3,p8),seller(
      o3,p9),publication_date(o3,p10),available_quantity(o3,p11),Author(o4),
      Publisher(o5),have(o1,o2),movie(o1,p12),id(o2,p13),based_on(o2,o3),movie(o3,
      p14),wrote(o4,o3),id(o4,p15),published(o5,o3),id(o5,p16),publisher(o3,p17)
```

Listing 5.4 – User-centric View of the service *getBook*, representing the Steiner tree 2



(a) Top 1 with a matching score = 0.857

(b) Top 2 with a matching score = 0.857

(c) Top 3 with a matching score = 0.857

Figure 5.11 – Generated Steiner Trees

```
VgetBook: Publisher(o1),id(o1,p1),Author(o2),Movie(o3),rating(o3,p2),title(o3,p3
    ),Award(o4),Seller(o5),location(o5,p4),Book(o6),isbn10(o6,p5),isbn13(o6,p6),
    price(o6,p7),author(o6,p8),seller(o6,p9),available_quantity(o6,p10),
    publication_date(o6,p11),published(o1,o6),publisher(o6,p12),wrote(o2,o6),id(
    o2,p13),took(o3,o4),awards(o3,p14),id(o4,p15),given_for(o4,o6),award(o6,p16)
    ,have(o5,o6),id(o5,p17)
```

Listing 5.5 – User-centric View of the service *getBook*, representing the Steiner tree 3

Remark 1: There are several ways to connect a set of nodes $V_i \subseteq V_s$, however our algorithm ensures that all leaf nodes of the discovered tree represent different V_i (and that V_{op} are Steiner nodes). To do so, Algorithm 11 finds different Steiner trees connecting a given node V_i and at least a node V_{op} such that V_i remains a leaf node and V_{op} represents either the root or a Steiner point. Then, at a given height h=1, . . . , H , the algorithm can discover a possible merger that can eventually represent the service view.

5.5 Experiments

To provide a basis for evaluating the quality of automatic services' views specification, we conducted a number of experiments using the results of different matching tools. Next, we propose metrics for measuring the performance of the *views generator* and justify the rationale behind our choice. Then, we will report on some of the significant results.

5.5.1 Performance Metrics

Comparing the data instances retrieved using the automatically generated services' views with the real (relevant) answers satisfying the user data queries can be used to evaluate the query reformulation quality[16] as shown in Figure 5.12. In particular, the set of automatically obtained answers is comprised of B, the data of interest or the true positives, and C, the false positives. False negatives (A) represent the requested data but not retrieved using the automatically generated services' views, while false positives (C) represent the set of irrelevant data which has been retrieved. Finally, true negatives (D) represent the data the user is not interested in and which has been correctly discarded by the query reformulations. Intuitively, false negatives and false positives reduce the views quality.

To evaluate the performance of the *views generator*, two objective measures from the information retrieval field, *Precision* and *Recall*, can be computed [20, 52][17] based on the cardinality of the sets described above. In a first set of experiments, we only consider the top-1 views automatically generated by our algorithm for the data services in the Service Lake, execute the corresponding web calls, and we finally examine the obtained results as follows:

Precision: is defined as the percentage of relevant instances among the retrieved data. It depicts the probability that the services' views automatically defined by our *views generator* return the information of interest to the user (the data requested in the data queries).

$$precision = \frac{|\{retrieved\,data\} \cap \{requested\,data\}|}{|\{retrieved\,data\}|} = \frac{|B|}{|B| + |C|} \tag{5.9}$$

[16] Query reformulation refers to the process of rewriting the service description over the user data source schema

[17] In [20], Precision and Recall are called soundness and completeness, respectively

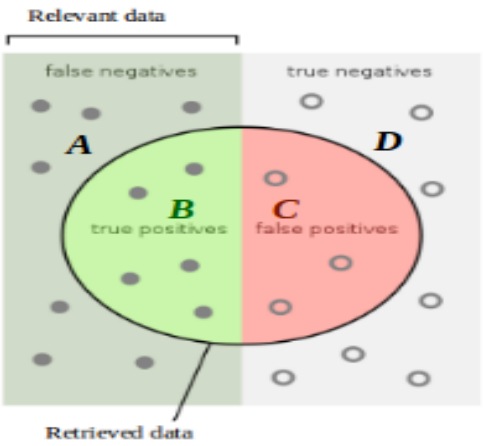


Figure 5.12 – Comparison between Requested and Retrieved Data Using the Automatically Generated Services' Views

Recall: is defined as the fraction of relevant data that have been retrieved over the requested data. In other terms, it depicts the probability that a requested information can be properly obtained using the selected service view.

$$recall = \frac{|\{requested\,data\} \cap \{retrieved\,data\}|}{|\{requested\,data\}|} = \frac{|B|}{|B| + |A|} \tag{5.10}$$

In the ideal case, when all and only the data of interest is returned (no false negatives and false positives are returned), we have Precision = Recall = 1. However, neither Precision nor Recall alone can efficiently assess the views quality. In particular, Recall can easily be maximized at the expense of a poor Precision by returning as many information as possible independently of its relevance, e.g. movies' titles instead of books' titles. On the other side, a high Precision can be achieved at the expense of a poor Recall by returning only few but correct answers. Hence it is necessary to consider both measures or a combined measure such as:

F-measure: is the weighted average of Precision and Recall, defined as the harmonic mean of both values and formulated as follows:

$$F - measure = 2 * \frac{precision * recall}{precision + recall} \tag{5.11}$$

Overall: represents a combined measure introduced in the context of schema matching, specifically to quantify the post-match effort needed to improve the

results of schema matching tools [55][18]. For the same purpose, we use the overall measure in our context to quantify the post-effort needed for improving the generated services' views quality which depends in a large part on the post-match effort needed to improve the matching results before proceeding to the views definition. The greater the overall value is, the less post-match effort the user has to provide.

$$Overall = recall * (2 - \frac{1}{precision}) = \frac{|A| + |C|}{|A| + |B|} = \frac{|B| - |C|}{|A| + |B|} \tag{5.12}$$

The notion of *Overall* only makes sense if precision is not less than 0.5 [31], i.e at least half of the returned answers are correct. Otherwise, we get a negative overall. Indeed, if more than a half of the obtained answers are not relevant, it would take the user more effort to remove the irrelevant data that has been integrated in the local database, look for the desired answers and integrate it in the database of interest than manually enriching the local dataset from scratch. The best overall 1.0 is achieved when both precision and recall are equal to 1.0.

In a second series of experiment, we go a step further to evaluate the impact of k, the number of top considered service views, on the outcome in terms of precision, recall, f-measure and overall values. We test our algorithm using the same data services that we used in the previous experiments. We vary k from 1, 3,... to 15, and report our results in Figure 5.12. Our objective is to maximize recall, f-measure and overall values without letting precision drop below 0.5.

5.5.2 Baseline Results & Discussion

The entire evaluation consisted of the generation of views over 100 data services from different domains such we investigated different matching methods and different matchers for deriving semantic mappings between the user data source schema and the services' descriptions. The quality measures *Precision*, *Recall*, *F-measure* and *Overall* were first determined for top-1 services' views and the results are summarized in Table 5.3.

The results show that the use of ground truth matching for the generation of service views significantly outperforms the use of approximate matches computed using different matching tools, with an averaged precision of 100%, a recall of 82%, an f-measure of 90% and an overall of 81%.

[18]Overall has been introduced in [55] under the name Accuracy

Matching Method	Precision	Recall	F-measure	Overall
Manual	1	0.82	0.90	0.81
COMA++ (NodesPath matcher)	0.66	0.42	0.78	0.20
COMA++(AllContext matcher)	0.93	0.66	0.80	0.61
COMA++ (NodesName matcher)	0.70	0.46	0.80	0.26

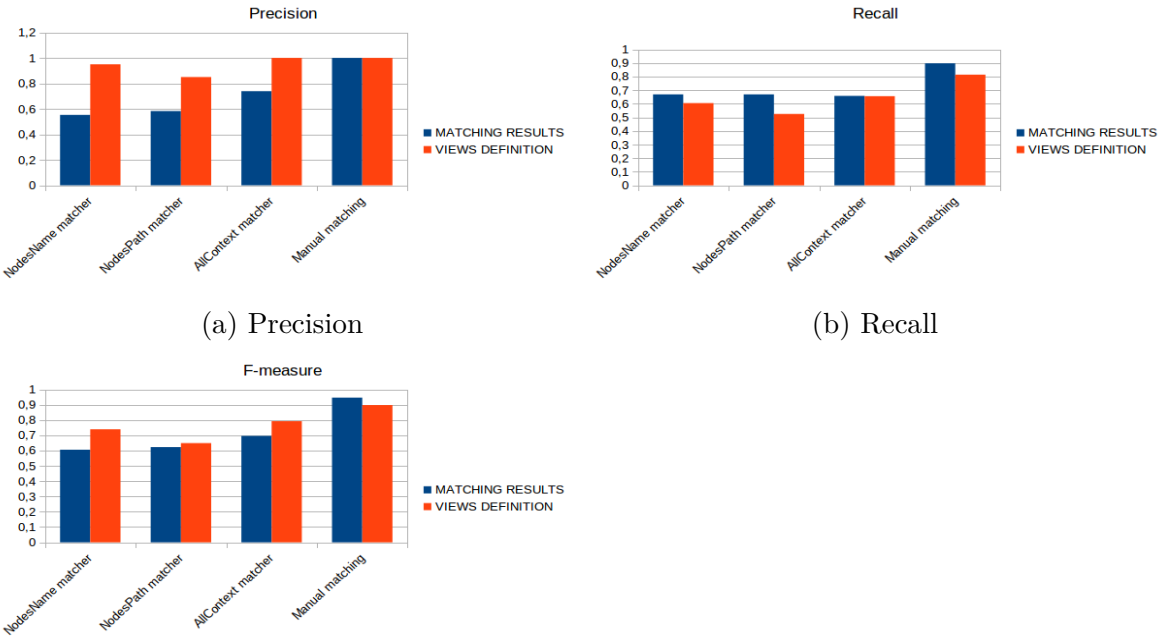
Table 5.3 – Average Precision, Recall and F-measure for Top-1 Services' Views Generated Based on Different Matching Results

Noteworthy is that it not always possible to find a matching partner for all services' parameters. The reason is that none of its correspondences exist in the database against which users' queries are posed, and thus obtaining a recall less than 1.0 even when using ground truth matchings. Given a 100% of precision, the overall is the same than recall. This confirms that overall is more pessimistic than f-measure in this case (81% against 90%).

We also notice that the reformulations of services' descriptions based on the *COMA++ AllContext* matcher results achieves a high precision of 93%. The reason is that the *AllContext* matcher identifies and matches all contexts by considering for a shared element all paths (sequences of nodes) from the root to the element, and it then identifies correct matchings [33]. On the other hand, both the *NodesName* and the *NodesPath* matchers produce the worst results. The results obtained based on the *NodesName* matcher results slightly achieves a better precision and recall than those results derived using the *NodesPath* matcher results (46% against 42% of recall and 70% against 66% of precision). We also notice that the small difference between the *AllContext* matcher and the two other COMA++ matchers (20% of recall and precision) involves a large gap with the overall (almost 40%). On the contrary, we notice a close f-measure for all the three different COMA++ matcher.

To conclude, the obtained results demonstrate the impact of the matching choices on the quality of the automatic specification of services' views. The more accurate the matching results are, the higher precision, recall, f-measure and overall values we get (see Figure 5.13).

The achieved results also vary when the number of nodes needed for the reformulation of services' descriptions varies. This number reflects the number of join relations that must be involved in the service view. The more the matches of services' parameters are scattered in different nodes in the schema graph, the worst(smaller) the precision is. The reason is that the number of nodes to be explored for the description of services' views is greater, thus increasing the number



(a) Precision

(b) Recall

(c) F-measure

Figure 5.13 – Matching Results versus Views Definition Results

of possible Steiner trees connecting those nodes.

In our second series of experiments, we varied k from 1 to 15. Then, we study its impact on all of the evaluation metrics described above in Figure 5.12. As expected, the average precision is always equal to 1 when using ground truth matchings, while the average recall, f-measure and overall decreases when k increases from 1 to 15. However, we notice a different behavior when using automatic schema matching results. The average precision, recall, f-measure and overall curves change once for the better and another for the worst when varying k. In particular, the quality of the views generated based on the matching results computed using the *allContext* matcher is much better than the other matchers' results quality and achieves its maximum when k=1. This mainly reflects the matching quality of the performed matcher such as the *allContext* matcher outperforms the other matchers in providing correct matchings. We also notice that the best quality results are not necessarily achieved at k=1 when the *NodesName* or the *NodesPath* matcher are applied to derive the semantic correspondences between the user schema and the services' parameters. The reason is that these matchers do not consider all the matching context and therefore they do not associate different matching scores between candidate correspondences since they are expressed similarly, which resulted in a large number of false positives. Hence, a

poor overall is achieved (less than 0.5).

In conclusion, our results indicate that our approach can perform reasonably well and output correct service views when ground truth matches are provided. Also, the use of the matching results derived by the *AllContext* COMA++ matcher achieves a good quality views (overall $>$ 60 %). In a nutshell, our approach is agnostic to how correspondences are created, but is cognizant that the matchings automatically computed are often incomplete, and sometimes incorrect. Despite many efforts in research and industry, even the best schema matching algorithms do not output 100% correct matching results, especially those fully-automatic algorithms where there is no human designer in the loop. Since the matching results largely influence the accuracy of the generated views, all suggestions are subject to user feedback for confirmation or correction.

5.6 Concluding Remarks

In this chapter, we have presented an automatic approach for defining data service views over the concepts of a given data source. We therefore studied finding top-k minimum cost group Steiner trees. The top-k found trees represent the best-matching service views of a given data service. The final choice of the most appropriate service view is done by a data expert. This approach is implemented as a prototype and validated by experiments using different real data services. The main motivation behind this work is the fact that despite many composition approaches that have been developed so far, the definition of service views has traditionally been carried out manually by experts when looking for services. To the best of our knowledge, there is no previous work that has dealt with the automation of the services' views definition.

Chapter 6

A Model-Driven Framework for the Modeling and the Description of Data-as-a-Service to Assist Service Selection and Composition

Contents

In the previous chapter, we have presented a new approach for automating the definition of services' views expressed in terms of vocabularies automatically derived from the schema of the user database under the assumption that services' descriptions are complete. The generated views would be significantly more useful if it were expressed in terms of commonly used vocabularies, a domain ontology for instance. Furthermore, most data providers in the cloud describe their services' capabilities in HTML pages of their websites. Thus, human efforts are always needed to read and to convert the HTML descriptions into a standard machine readable model. With no doubt, reading all the different DaaSs' documentation each time we want to add new DaaS services to the Service Lake is not the best alternative. In such context, our main aim of this work is to make DaaS descriptions complete and machine readable in order to assist automatic service selection and composition (see Chapter 4) .

We present in this chapter *MoDaaS*, a Model-Driven framework for the modeling and the description of data services and DaaS services in particular. This is an intuitive abstraction, based on MDE standards, enabling DaaS providers to model the services they propose according to a domain ontology and to automatically generate RDF views over the concepts of the underlying ontology. Our work can be seen as a complementary approach to many existing approaches for the discovery and the composition of data providing services. This chapter is organized as follows. In Section 6.1, we first describe briefly the background of Model Driven Engineering (MDE). Then, we discuss how to apply MDE techniques to assist service selection and composition. Section 6.2 presents MoDaaS, our DaaS modeling framework, the whole modeling process giving a detailed description of each step. After that, we describe the implementation and we showcase the working of our solution using a case study in Section 6.3, and finally we conclude the chapter in Section 6.4.

6.1 Model Driven Engineering for the Cloud

In this section, we first review some important MDE notions. Then, we discuss how MDE techniques can be used to assist service selection and composition.

6.1.1 MDE, Model Driven Engineering

MDE is becoming an emergent software engineering paradigm to specify, develop and maintain software systems where models are the primary artifact of the engineering process and are used, for instance, to (semi)automatically generate the implementation of the final software system [10].

According to the Object Management Group (OMG), MDE is a specific approach to software engineering that defines a theoretical framework for generating a code from models using successive model transformations. The main goal of this approach is to separate the business side of a system from its implementation. The business model of a system can therefore drive its implementations on different platforms. In this way, MDE aims to raise the level of abstraction in program specification and increase automation in program development, and thus, we can expect to obtain better coherence between implementation and interoperability.

In particular, MDE is based on two key elements: meta-modeling and model transformations. Meta-modeling defines the structure and semantics of domain models, while transformations ensures the automatic model manipulation for different purposes, like refactoring, model querying, code generation, language mappings and conversions between technological spaces. The best-known MDE initiative is the MDA proposed by the OMG[19].

6.1.2 MDA, Model Driven Architecture

Model Driven Architecture (MDA) is a model driven approach to software development governed by the OMG, where models abstract the business functionality of the software system from its implementation on specific platforms. More specifically, it aims to separate the application structure (PIM, Platform Independent Model) from its functionality (PSM, Platform Specific Model). The mapping between these models is realized by model transformation, either the model-to-model transformation or model-to-text transformation.

Definition 5 *A transformation is defined as the process of the automatic generation of a target model from a source model, according to a set of defined rules.*

All transformations are based on the abstract syntax, the *meta-model*. In a model-to-model transformation, the source model is transformed to another model as

[19]http://www.omg.org/mda/

illustrated in Figure 6.1. Whereas, in a model-to-text transformation, the model is transformed to text (a source code or any other kind of text).

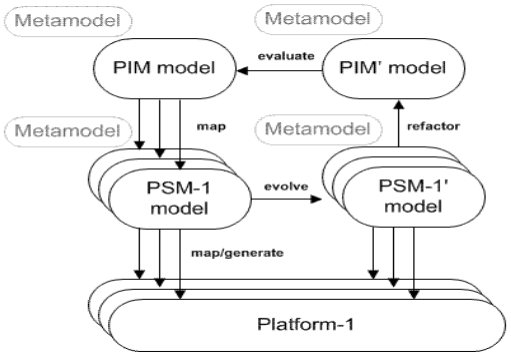


Figure 6.1 – Process of the Model-Driven Architecture

6.1.3 MDA for the Modeling of DaaS

As mentioned above, MDA techniques can facilitate and (semi-)automate the modeling and the development of new services and applications. Several approaches have already explored this possibility for different purposes [23, 58, 63, 10, 60]. Yet, there is not a consensus on the right set of models, languages, model transformations for the model-driven modeling and development of data services despite the numerous advantages and benefits that MDA techniques can bring to make the most of data services. Follows are the most important:

- *Construction of a generic and complete model for the description of DaaS:* Service consumers need to distinguish services based on their functional and non-functional criteria to make the most appropriate choice among a number of DaaSs providing the same information. Therefore, we emphasize the need for a modeling framework to semi-automatically create DaaS models that define the capabilities of DaaS services, including functional and non functional properties, and to make DaaS descriptions complete and machine readable. MDA promotes the creation of machine-readable, highly abstract DaaS models that are developed independently from its implementation technology. Thereafter, the created DaaS models can be accessed repeatedly and automatically processed by tools.
- *Overcoming data heterogeneity:* One of the major issues faced during the selection and the composition of data services is the possible heterogeneities that can occur between the user data query and the service description

or among different services' descriptions in a composition. Considerable attention has been focused recently on MDE as an alternative solution to overcome heterogeneities' concerns. In our context, enabling DaaS providers to integrate semantic information about the services and the data they provide by modeling their functional capabilities according to a shared ontology within a model-driven framework can help to bridge the different knowledge representations. Thereby, DaaS capabilities and their associated data concerns can be easily processed by service selection and composition tools.

- *Reusability:* Using model transformations, the same DaaS model can be automatically transformed into different reformulations and thereafter it can be reused to generate views over different ontologies, and uunder different formats (RDF views, Conjunctive queries, etc.). This constitutes a strong benefit since it avoids the time waste in manual implementing views.

As far as we know, only ODaaS, a model-driven initiative has already explored this possibility [74] in the literature. Our work complements these efforts by providing an integrated framework to automatically define services' views over a mediated ontology given their descriptions. The key idea is to define a higher description level on top of the traditional service descriptions in WSDL, OWL-S, etc, among which DaaS providers define their DaaS capabilities. More specifically, we are interested in creating and exploring well-established models to represent all the concerns of DaaSs, including data quality concerns; in particular, (1) a meta-model that captures all the (main) concerns associated with DaaS services and the provided data, (2) a modeling tool which provides an intuitive user interface allowing DaaS providers to define and model their DaaS services according to a shared ontology, and finally (3) a view generator which establish mappings between the data provided by the DaaS and the domain ontology. In the following, we will present MoDaaS, its building blocks and how they tackle the problems identified above.

6.2 MoDaaS for the Modeling and the Description of DaaS

Our goal is to provide a complete solution that assists DaaS providers to describe their services capabilities and properties. In this section, we present more details about the MoDaaS architecture, giving the main components and the main steps required for modeling and annotating DaaS services.

6.2.1 Overall Architecture

Figure 6.2 shows MoDaaS architecture. It includes three main modeling tools: *DaaSMetaModel*, *MoDaaS Editor* and the *View Generator*. We have used Eclipse Modeling Framework to develop the meta-model and the generative capabilities within our framework. Actually, both MoDaaSEditor and ViewGenerator use DaaSMetaModel in order to guarantee the creation of only valid models and the generation of relevant service views.

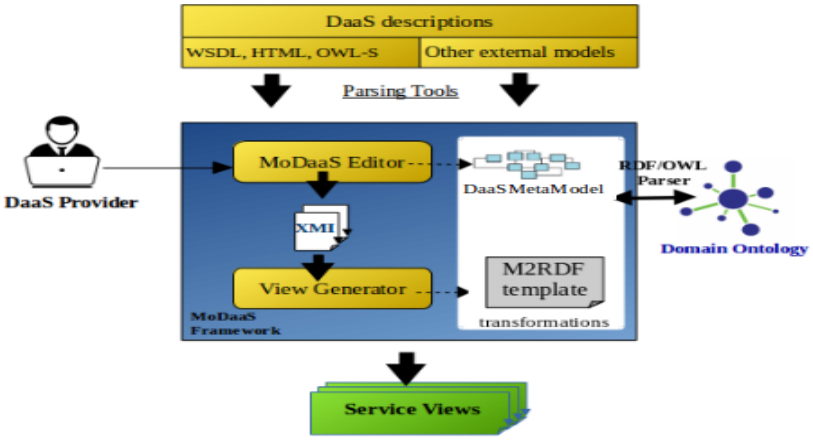


Figure 6.2 – Overall Architecture of the MoDaaS Framework

Following MDA practices, the modeling and the annotation of a DaaS service is achieved in three-step process:
First, DaaS providers describe their service capabilities and properties through the creation of a new DaaS model under MoDaaS Editor. The resulting DaaS model is saved as an XMI description consisting of several nodes. Each node may correspond to a DaaS provider, a DaaS operation, an I/O parameter or to a service property. Moreover, providers can check for the validity of their models during their creation.

Second, providers need to annotate their DaaS models through matching the different input and output parameters with the shared ontology concepts. This is an important step to deal with data heterogeneity. DaaS providers can also introduce new ontologies of their choice to the system (i.e., by specifying the ontology url/path). For this purpose, we implemented a RDF/OWL parser to extract the concepts of the introduced ontology and to dynamically enrich the MoDaaS Editor. Thereafter new concepts are introduced in the editor within the I/O annotation property, enabling providers to define their services through a conceptual representation of the domain of interest provided in terms of the

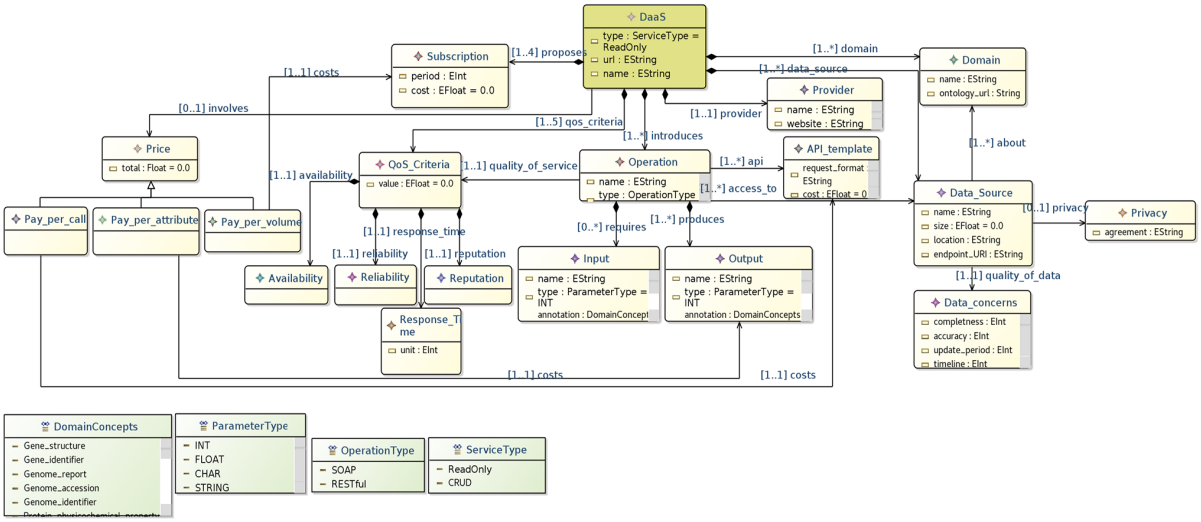


Figure 6.3 – DaaSMetaModel

introduced ontology, to which DaaSs are mapped.

Finally, the resulting XMI file can be leveraged to define service views over the shared ontology. In doing so, the M2RDF transformation pattern is executed to define the RDF view corresponding to the DaaS model.

6.2.2 DaaSMetaModel: a Meta-model for the Modeling of DaaS Concerns

MoDaaS is built on top of the meta-model DaaSMetaModel. This latter provides an extended DaaS description model for ensuring interoperability between different DaaS services. In particular, it captures all the main DaaS concerns and aspects that need to be addressed in its description, and being represented as extensible and customizable classes in the model. *DaaSMetaModel* is depicted in Figure 6.3. In the following, we describe the different meta-model components and their responsibilities:

- **DaaS:** is the main class of our model. It is a special type of cloud services that provides data on demand. According to [78], DaaSs can be categorized into:
  - **Read-only DaaS:** is a data service which provides data based on existing data sources in the cloud.
  - **CRUD DaaS:** is a data service which allows the consumer to create,

retrieve, update and delete data. They typically provide just a storage capability and it is up to consumers to define their own data schema and/or to publish their data.

- **Provider:** represents the cloud provider serving data through the cloud.
- **Operation:** is a function that processes a set of inputs and results in a set of outputs, or associates arguments (inputs) with values (outputs).
- **Input:** input parameter can be of any data type, and can be set as a required parameter on the service. If it is set as required, the service can not be run if the parameter is not provided.
- **Output:** it represents the information returned by a DaaS service. It can be of any data type.
- **QoS criteria:** is the description or measurement of the overall performance of a service DaaS. To quantitatively measure quality of service, several related aspects of the network service are often considered, such as :
  - **Reputation:** is a measure of the service trustworthiness. It mainly depends on end user's experiences of using a DaaS.
  - **Reliability:** is the probability that a request is correctly responded within the maximum expected time frame indicated in the DaaS description.
  - **Availability:** is the percentage of time that a DaaS service is operating.
  - **Response time:** is the expected delay in seconds between the moment when the request is sent and the moment when results are received.
- **Pricing model:** aims to represent the cost information when a DaaS web call is executed. Different payment models, describing how to charge consumers for using DaaS services, are proposed such as:
  - **Payment on access (per call):** DaaS users are charged every time they call a DaaS API to retrieve data. The API usage fee is specified in the API description.
  - **Payment on data type and data size (per attribute):** users are charged according to the type and the size of the requested data, such as only unit prices are described.
  - **Payment on plan (for volume):** users subscribe for data usage in a period (e.g., a week, a month, or a year) and only pay once in this period with or without limitations for how frequent they access data and how much data they retrieve from DaaSs.

- **Data source:** represents the data source a DaaS is accessing to, such it is stored in the cloud.
- **Data concerns:** our data model characterizes also the datasets a DaaS access to, with specific attributes including accuracy, completeness, update periode, and timeliness, which constitute the focus of the majority of data consumers:
  - ***Completeness:*** is the degree to which a given data collection includes data describing the corresponding set of real-world objects. The completeness implicitly compares the information available in the source with what holds in the world.
  - ***Accuracy:*** is the extent to which data are correct, reliable and certified. In other terms, the data provided by a DaaS is said accurate when the values returned by the service correspond to real-world values.
  - **Update period:** represents the delay between two major changes in the data source, set to daily by default.
  - **Timeline:** describes the lifetime of the data.
- **API template:** Data providers publish global APIs to make data available. A data request for a given DaaS service is expressed in the same format; given a set of values for the input parameters, the service execution generates values for the output parameters of the service.
- **Domain:** incorporating semantic information about DaaS services, besides the typical information on service inputs and outputs can help selecting the relevant DaaSs (that provides the data a user is requesting, without human assistance), given a data query.

This is the core model which was extended incrementally by adding new and derived classes to manage all the DaaS and providers concerns. An important extension is the introduction of the "Domain" class, enabling the selection of a domain ontology. Each input and output parameter must be then matched to its correspondence from the chosen ontology. Such information is stored in the attribute ***'annotation'***. The re-use factor can be further increased through the introduction of semantic information to DaaS models.

6.2.3 MoDaaS Editor & Semantic Annotation of DaaS

The model developed above provides the necessary theoretical background to represent DaaS capabilities and properties. In the following, we explain

in details how DaaS providers can model and annotate their services within MoDaaS.

MoDaaS Editor provides an intuitive user interface that allows DaaS providers to define their services and the data they provide, through a simple graphical modeling tool. A screenshot is shown in Figure 6.4. It is mainly composed of three layouts:

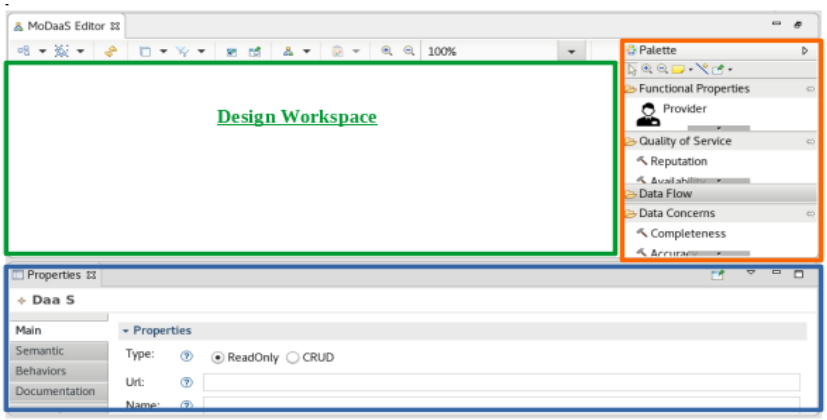


Figure 6.4 – MoDaaS Editor

- **Design Workspace:** Here, users can define new DaaSs by designing and validating the corresponding DaaS model. The essential building blocks of a DaaS model, such as Operation, Input, Output, API template, etc., can be dragged and dropped from a tools palette.
- **Palette:** It contains all the elements that can be used to create a DaaS model, in other terms to define and describe a DaaS service (input/ output parameters, data source, data concerns, etc.).
- **Configuration Tabs:** each tab opens a view that displays the properties of the selected element in the design workspace (e.g., the name of an input/output parameter and its corresponding data type, the name and the size of a data source, etc.). All of these functional and non-functional properties can be specified and edited.

The creation of a new DaaS model consists of defining the provider, the input and output parameters, the QoS values, etc. Such information must be specified in the configuration tab corresponding to each class. DaaS providers must also integrate semantic information about their services by modeling their capabilities according to a shared ontology in MoDaaS. As we said earlier, it is also possible to introduce new ontologies in MoDaaS. The

semantic annotation of DaaS services is performed manually by selecting the corresponding concept from the chosen ontology for each service I/O parameter (cf. Figure 6.7).

6.2.4 Automatic Generation of RDF Views

The main idea behind service views is to deal with the possible semantic and structural heterogeneities that may exist between the data formats and types returned by different DaaS services. As a second contribution, our model-driven framework provides a set of model-to-text transformations to be automatically executed in order to generate service views from input DaaS models. In the literature, most service selection and composition approaches [81], [18], [7], explicitly regard DaaS as Parametrized RDF Views (PRVs) over a mediated ontology to capture their semantics in a declarative way. Thus, we propose in this work a transformation template, we call ***M2RDF transformation pattern***, which given an input DaaS model, generates the corresponding RDF view over a domain ontology, initially shared in the MoDaaS framework.

Using the annotations of the form described in the previous section, M2RDF extracts the service view from the DaaS description file (xmi file) as follows:
Input and output parameters are declaratively represented based on concepts and relations that are semantically defined in the domain ontology and selected by the DaaS provider during the service annotation. We recall that a data service requires a particular set of inputs (i.e., the parameter values) in order to retrieve a particular set of outputs and thus outputs cannot be retrieved unless inputs are bound. Therefore, it is also necessary to indicate in the RDF views to generate, which parameters are inputs and which parameters are outputs. Thereafter, each view is characterized by an access pattern, specifying whether a parameter is input or output such as input and output variables are prefixed with the symbols \$ and ?, respectively. Formally, an RDF view of a DaaS service S_i over a domain ontology is a predicate of the form: $S_i(\$\overline{\mathrm{X_i}}, ?\overline{\mathrm{Y_i}}) =< \$\overline{\mathrm{X_i}}, ?\overline{\mathrm{Y_i}}, R(\overline{\mathrm{X_i}}, \overline{\mathrm{Y_i}}) >$, where:

- $\overline{X_i}$ and $\overline{Y_i}$ are the sets of input and output parameters of S_i, respectively.
- $\mathrm{R}(\overline{\mathrm{X_i}}, \overline{\mathrm{Y_i}})$ represents the semantic relationship between input and output variables using RDF triples of the form (subject,property,object). These RDF triples are extracted from the .owl file representing the domain ontology.

Using RDF Views expressed in terms of a shared ontology has several advantages.

(1) The use of views over a mediated schema known from traditional database integration area allows us to overcome the problem of data and schemas heterogeneity. (2) RDF Views can be used to figure out what possible data can be returned by a given data service. (3) Expressing the mediated schema in terms of the OWL ontology, which is independent of the local database technology, allows us to treat different data sources in the same fashion (e.g., relational databases, RDF stores, etc.). (4) Describing the data service as an RDF View independent from the real physical schema behind allows to tackle the heterogeneity issues faced during service selection and composition. This is explained in the following section.

6.2.5 Assistance to DaaS Selection & Composition

Semantic, syntactic and structural heterogeneities could exist between the keywords in the data query and the I/O parameters of data services, due to the lack of consistent semantic annotation. For this purpose, most service selection and composition approaches regard DaaSs as PRVs over a mediated ontology to cope with possible heterogeneities. This assumption requires the definition of service views beforehand, which has traditionally been carried out manually by experts when looking for services. This is a complex task that requires considerable effort, especially when considering a vast amount of services. By means of automatically generating services' views using *MoDaaS*, users are no longer concerned with finding and manually orchestrating the relevant services. They only need to specify their data queries over a domain ontology from *MoDaaS* (stored internally in the system repository) and then the *Service Composition Engine* of *EuDaSL* will find and orchestrate the DP services needed in answering the query in a transparent and integrated fashion. Recall that answering a data query may also require investigating combinations of multiple and heterogeneous DaaSs. The same way, RDF views helps capturing all the possible and more certain interactions between the component services, resulting valid and executable compositions.

6.3 Validation

To evaluate and assess the value of the proposed approach, we implemented MoDaaS on top of the Eclipse Modeling Framework (EMF). In the following, we first introduce the implementation details. Then, we present a case study in biology to demonstrate how our editor can be easily used to model and semantically

annotate DaaS services. For this end, we considered three different services from Amazon[20], EMBL-EBI[21] and from BioComputingUP[22].

6.3.1 Implementation

EMF comprises a number of model-driven development capabilities such as the modeling, the inter-model transformations, and code generation. More precisely, it allows the definition of models and metamodels by means of the Ecore tools. We created the meta-model *DaaSMetaModel* using the Ecore meta-metamodel, and based on which, we built the editor *MoDaaS* enabling DaaS providers to create and to manage DaaS models w.r.t DaaSMetaModel. Further, EMF provides model-to-text transformations support through Acceleo. We used this latter to implement the views generator M2RDF. The MoDaaS implementation process and its building blocks are represented in Figure 6.5.

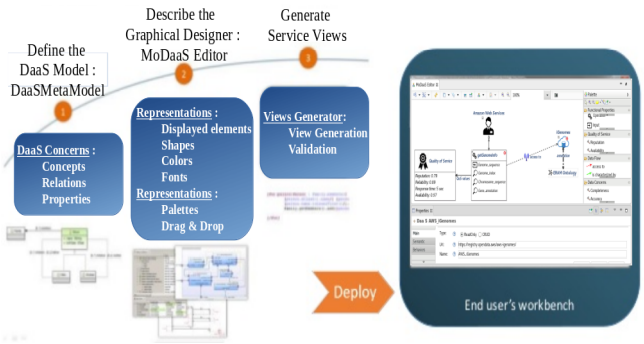


Figure 6.5 – MoDaaS Implementation Process

6.3.2 Case Studies

As a first case study, we consider the DaaS *AWS-iGenomes*[23] which represents a collection of reference sequences and annotation files for commonly analyzed organisms. This information is made accessible using the service *getGenomeInfo*. Our aim is to model and semantically annotate it using MoDaaS. In our current prototype, we propose the well-known ontology *EDAM* according to which we want to annotate the iGenomes capabilities. EDAM is a comprehensive ontology of well-established, familiar concepts that are prevalent within bioinformatics and computational biology, including types of data and data identifiers, data formats,

[20]https://registry.opendata.aws/
[21]https://www.ebi.ac.uk/
[22]http://protein.bio.unipd.it/
[23]https://registry.opendata.aws/aws-igenomes/

operations and topics [45]. We provide in the following a step-by-step explanation of the DaaS modeling process in MoDaaS:

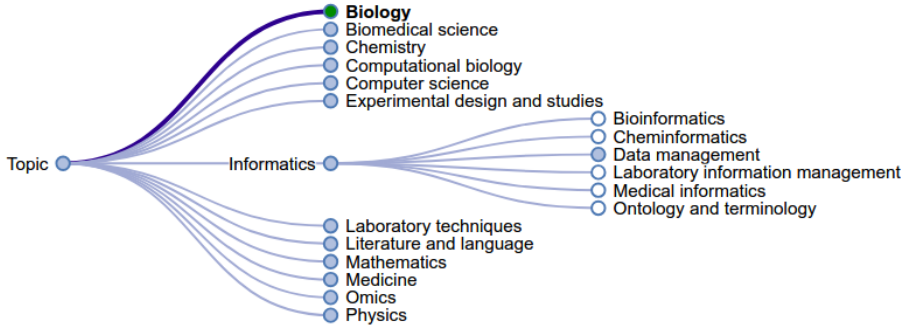


Figure 6.6 – EDAM Ontology

6.3.2.1 Model Design

The first step was the creation of a DaaS model describing all the capabilities and the properties of the service *getGenomeInfo*. We defined at the top of the model an instance of the DaaS_provider metaclass representing the provider AWS. We then created and defined an input "Genome_sequence" and three outputs , "Genome_index", "Chromosome_sequence" and "Gene_annotation" within the service operation. Each parameter has been matched to a concept from the chosen domain ontology, the EDAM ontology in this case. Actually, the semantic annotation of an operation parameter consists of selecting the corresponding concept, matching the parameter, from the set of proposed concepts within the property *annotation* as shown in Figure6.7. This set can be automatically and dynamically enriched each time a new ontology is introduced to the MoDaaS editor. Finally, as each DaaS has its own properties such as the response time, the reputation, the availability, etc. , we defined these values in the configuration tab corresponding to each property.
Note that the validation of DaaS models can be performed during their creation. Here, in Figure 6.7 we notice the appearance of two different errors. More details about the errors may be shown by clicking on the red cross. For example, the error depicted in the figure comes from the fact that we did not define the API template. This is very useful for figuring out and avoiding model errors. Once all errors are fixed, the provider can launch the generation of service views process.

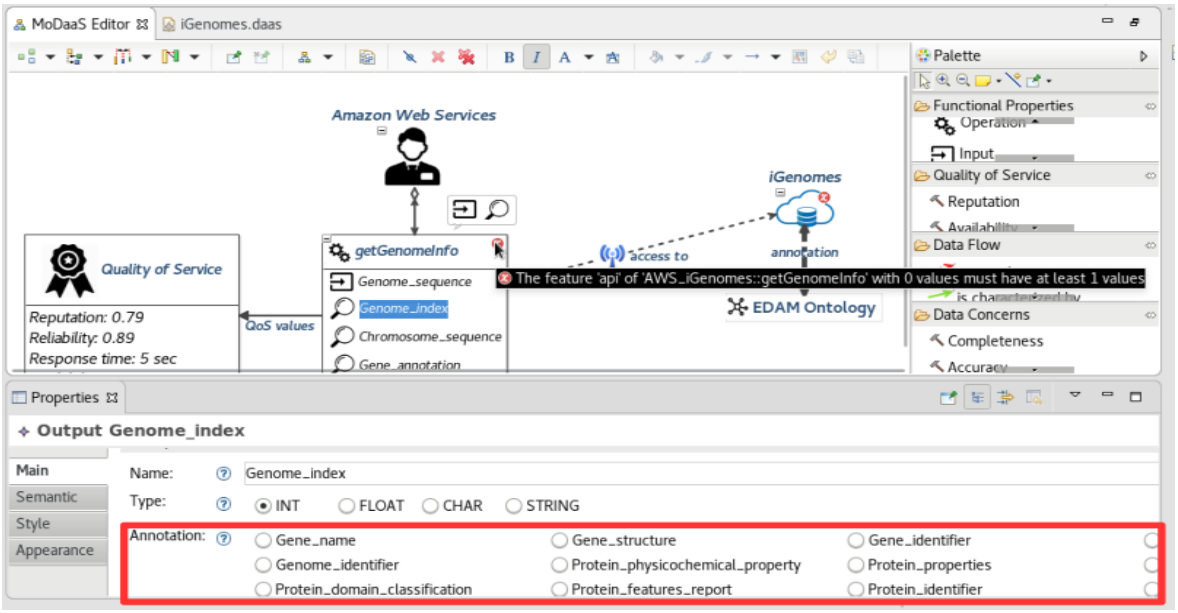


Figure 6.7 – DaaS Modeling and Annotation according to the EDAM Ontology

6.3.2.2 Service Views Generation

The generated RDF view is depicted in Listing 6.1. This is the outcome of the execution of the transformation pattern M2RDF on the DaaS model *"iGenomes.daas"*, created for the service *getGenomeInfo* (cf. Figure 6.8). For the sake of the simplicity, we just represent the RDF triples representing the corresponding ontology concepts and not the semantic constraints.

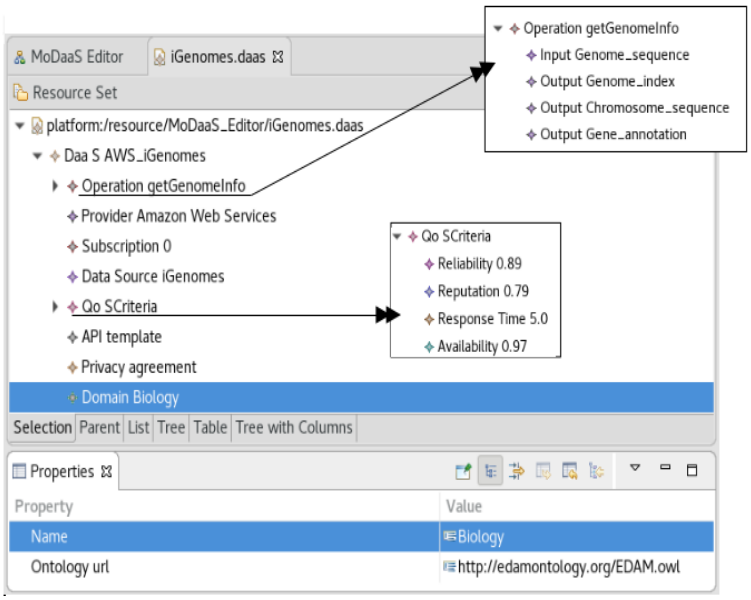


Figure 6.8 – EMF Tree View and Property Sheet for the Created DaaS Model (*.daas)

```
getGenomeInfo($genome_sequence, ?genome_index, ?Chromosome_sequence, ?
    gene_annotation'):
$<http://edamontology.org/data_2909> <http://www.w3.org/2000/01/rdf-schema#label
    > "Organism name"
?<http://edamontology.org/data_3210> <http://www.w3.org/2000/01/rdf-schema#label
    > "Genome index"
?<http://edamontology.org/data_0919> <http://www.w3.org/2000/01/rdf-schema#label
    > "Chromosome report"
?<http://edamontology.org/data_0916> <http://www.w3.org/2000/01/rdf-schema#label
    > "Gene report"
```

Listing 6.1 – Generated RDF View for the service *getGenomeInfo*

The RDF triples defined in Listing 6.1 were originally extracted from the EDAM ontology. Given the xmi file(iGenomes.daas), the M2RDF template extracts the RDF triple representing each service parameter in the mediated ontology. For instance, the genome index is represented in the EDAM ontology using the RDF triple ***<http://edamontolo- gy.org/data_3210><ht tp://www.w3.org//rdf-schemalabel>"Genome index"***.

The same way, we generated the service view of *getProteinByName* (Listing 6.2) providing access to information stored in the DisProt database[24], a database of experimental evidences of disorder. This service returns the UniProt accession, the DisProt ID, the organism name, its taxonomy and the set of homologous entries, given a protein name.

```
getProteinByName($Protein_Name, ?DisProt_ID, ?UniProt_Accession, ?Organism, ?
    Taxonomy, ?Homologous_entries'):
$<http://edamontology.org/data_1009> <http://www.w3.org/2000/01/rdf-schema#label
    > "Protein name"
?<http://edamontology.org/data_2723> <http://www.w3.org/2000/01/rdf-schema#label
    > "Protein ID (DisProt)"
?<http://edamontology.org/data_3021> <http://www.w3.org/2000/01/rdf-schema#label
    > "UniProt accession"
?<http://edamontology.org/data_2909> <http://www.w3.org/2000/01/rdf-schema#label
    > "Organism name"
?<http://edamontology.org/data_1868> <http://www.w3.org/2000/01/rdf-schema#label
    > "Taxon"
?<http://edamontology.org/data_3148> <http://www.w3.org/2000/01/rdf-schema#label
    > "Gene family report"
```

Listing 6.2 – Generated RDF View for the service *getProteinByName*

As a third case study, we considered the Proteins REST API which provides access to key biological data from UniProt and from Large Scale Studies, including integrated protein and genome information [61]. Different services were proposed to enable data consumers to access the data on the fly. Listing 6.3 depicts the generated RDF view of the service *getUniProtEntryByAccession*, which given an UniProtKB accession number, returns some entry related information such as the protein name, the gene name, the organism name and the taxonomy.

[24]http://www.disprot.org/

```
getUniProtEntryByAccession($accession, ?name, ?Protein_name, ?Gene_name, ?
    Organism_name, ?Taxon'):
$<http://edamontology.org/data_3021> <http://www.w3.org/2000/01/rdf-schema#label
    > "UniProt accession"
?<http://edamontology.org/data_2291> <http://www.w3.org/2000/01/rdf-schema#label
    > "UniProt ID"
?<http://edamontology.org/data_1009> <http://www.w3.org/2000/01/rdf-schema#label
    > "Protein name"
?<http://edamontology.org/data_2299> <http://www.w3.org/2000/01/rdf-schema#label
    > "Gene name"
?<http://edamontology.org/data_2909> <http://www.w3.org/2000/01/rdf-schema#label
    > "Organism name"
?<http://edamontology.org/data_1868> <http://www.w3.org/2000/01/rdf-schema#label
    > "Taxon"
```

Listing 6.3 – Generated RDF View for the service *getUniProtEntryByAccession*

6.4 Concluding Remarks

This chapter presents a model-driven framework for the modeling and the description of DaaS. Our goal was to provide flexible means of modeling DaaS services and to allow their easy integration and invocation by data intensive analysis processes. We describe the data services provided by DaaS providers as views over a mediated domain ontology in order to deal with schemas heterogeneity. Specifically, the capabilities (the local schema) of each DaaS is mapped to concepts of a domain ontology, and its terms are used to define RDF service views. The generated service views are then stored in a repository and thus can be re-used and automatically processed by tools for answering data queries reformulated over the same ontology.

Chapter 7

Conclusion & Perspectives

Contents

In this concluding chapter, we first summarize the contributions of this thesis. Next, we discuss possible improvements on our ongoing works and some perspectives that we aim to realize at short and long terms.

7.1 Context & Contributions

This dissertation has advanced the state of the art of data integration by designing and implementing new solutions for enriching companies' data sources. In the light of recent advances in the field of web engineering, along with the increase number of data services providing access to timely and high-quality information, data services rapidly became the leading solution in providing valuable services to answer data queries on the fly. In this thesis, we make use of data services for enriching companies' data sources in order to provide complete and relevant answers to the user data queries. Although such enrichment task can be manually implemented, it can be a tedious, error-prone and challenging task to accomplish, especially when dealing with a large number of heterogeneous data sources.

The work presented in this dissertation lays ground for an active user-centric data integration approach, aiming at alleviating developers and data scientists tasks for enriching local data sources, and answering complex data queries. In particular, we made four main contributions that cover the user-centric data integration process as follows:

User-Schema Enrichment: Our first contribution, presented in Chapter 3 aims at identifying the missing data that is required for the processing of users' data queries but does not exist in the data sources they are interested in. Thereafter, it enriches the underlying schema with the determined missing concepts and associated attributes in order to enable the storage of new data when needed. This contribution has been published in the International Conference Business Information Systems (BIS 2017) [8].

Quality-Driven Service Selection and Composition for Query Answering: The objectives of our second contribution, presented in Chapter 4 are twofold. First, the selection of the relevant data services that can be used to leverage the missing information in user data sources and fulfilling the data queries he is interested in. Second, the selection of the best-quality query plans for the enrichment of the user data source, given a cost threshold specified by the user. In doing so, we elaborated a knapsack-based algorithm to select services that yield good quality results without exceeding a given cost threshold (time and monetary cost). The evaluation that we conducted showed the effectiveness of our solution. This contribution has been published in the IEEE International Conference on Web Services (IEEE ICWS 2018) [7]

Service Views Generation: Our third contribution, presented in Chapter 5 presents an automatic approach for defining data service views as conjunctive queries over the concepts of the user data source. We therefore studied finding top-k minimum cost group Steiner trees. The top-k found trees represent the best-matching service views of a given data service. This approach is implemented as a prototype and validated by experiments using a collection of real data services.

Model-Driven Framework for the Modeling and the Description of Data Services: MoDaaS, our fourth and final contribution, presented in Chap-

ter 6 presents a model-driven framework for the modeling and the description of data services, more particularly the Data-as-a-Service model. Our goal was to provide flexible means of modeling DaaS services and to allow their easy integration and invocation by data intensive analysis processes. We describe the data services provided by DaaS providers as views over a mediated domain ontology in order to deal with schemas heterogeneity. Specifically, the capabilities (the local schema) of each DaaS is mapped to concepts of a domain ontology, and its terms are used to define RDF service views. The generated service views are then stored in a repository and thus can be re-used and automatically processed by tools for answering data queries reformulated over the same ontology. This contribution has been published in the International Conference on Database and Expert Systems Applications (DEXA 2019) [9].

7.2 Open Issues & Future Directions

Our work still has a high potential for future improvements. We conclude our discussion by enumerating some explicit future research concerns.

7.2.1 Specification of Data Services' Views

In spite of the recent improvement in automatic matching tools, the matching process often remains a semi-automatic process. Indeed, after executing a tool on the schemas to match, the expert has to discard part of the proposed mappings and to look for the ones missed by the tool. This emphasises the need for implementing an interactive mode to capture the user feedback, in order to improve the matching quality and then the quality of the generated views. We mainly aim to provide two execution modes: automatic and interactive. In the first mode, the system takes the schemas, interacts with an automatic matching tool to compute the matches, without any user intervention. Whereas in the second mode, users can provide feedback during the matching process, and the system can selectively rerun the generation of service views based on the feedback. Furthermore, the schema matching techniques that have been used can be combined with other techniques to produce more accurate matching schema in query rewriting.

7.2.2 Selection and Composition of Relevant Data Services for Answering User Queries

More and more web services with similar and overlapping data but different QoS are available in the marketplace. Our service selection algorithm performs well on sets of 100 data services, however efficient selecting from a set with a larger number of services may bring some performance challenges. In this sense, we plan to update the algorithm 8 and make use of MapReduce to select and compose the appropriate services efficiently and equally from masses of services with the aim of reducing the response time and sharing the computational workload more fairly. As a matter of fact, MapReduce is a programming model and an associated implementation for processing large data sets in parallel on different clusters in a reliable and fault-tolerant manner, while hiding the complex underlying details of parallelization, distributed storage, load balancing and fault tolerance. Based on these, if we apply MapReduce to our work, it will improve the efficiency of selecting the appropriate services from masses of data services.

7.2.3 Modeling and Description of Data Services

As a future work, we plan to implement a new user interface, enabling users to model a composition of DaaS services, with generative tools to estimate the QoS values of the composition, and to automatically generate and execute the corresponding web calls. Actually, we are implementing the aggregation functions proposed in Chapter 4 for the estimation of QoS values, given a service composition. We will also be examining new techniques to generate data mapping workflows that map data from a specific type to another, more specifically from the type defined by the DaaS provider to a type in a domain ontology. This will help to handle the structural and the semantic heterogeneities that may exist between the data provided by different DaaSs in a service composition.

Bibliography

[1] Amazon data sets, http://aws.amazon.com/publicdatasets/.

[2] Enigma, https://www.enigma.com/public-data.

[3] Microsoft data market, https://datamarket.azure.com/.

[4] Oracle data cloud, https://www.oracle.com/fr/applications/customer-experience/data-cloud/.

[5] *2008 IEEE International Conference on Web Services (ICWS 2008), September 23-26, 2008, Beijing, China*. IEEE Computer Society, 2008.

[6] S. Adali and R. Emery. A uniform framework for integrating knowledge in heterogeneous knowledge systems. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 513–520, March 1995.

[7] Hiba Alili, Khalid Belhajjame, Rim Drira, Daniela Grigori, and Henda Hajjami Ben Ghézala. Quality based data integration for enriching user data sources in service lakes. In *2018 IEEE International Conference on Web Services, ICWS 2018, San Francisco, CA, USA, July 2-7, 2018*, pages 163–170, 2018.

[8] Hiba Alili, Khalid Belhajjame, Daniela Grigori, Rim Drira, and Henda Hajjami Ben Ghézala. On enriching user-centered data integration schemas in service lakes. In *Business Information Systems - 20th International Conference, BIS 2017, Poznan, Poland, June 28-30, 2017, Proceedings*, pages 3–15, 2017.

[9] Hiba Alili, Rim Drira, Khalid Belhajjame, Henda Hajjami Ben Ghézala, and Daniela Grigori. A model-driven framework for the modeling and the description of data-as-a-service to assist service selection and composition. In *Database and Expert Systems Applications - 30th International Conference, DEXA 2019, Linz, Austria, August 26-29, 2019, Proceedings, Part I*, pages 396–406, 2019.

[10] Hiba Alili, Rim Drira, and Henda Hajjami Ben Ghézala. Model driven framework for the configuration and the deployment of applications in the cloud.

In *2016 International Conference on Cloud Computing, GRIDs, and Virtualization, CLOUD COMPUTING 2016, Rome, Italy, March 20-24, 2016*, pages 61–68, 2016.

[11] Alexander A. Anisimov. Review of the data warehouse toolkit: the complete guide to dimensional modeling. *SIGMOD Record*, 32(3):101–102, 2003.

[12] Mahtab Arafati, Gaby G. Dagher, Benjamin C. M. Fung, and Patrick C. K. Hung. D-mash: A framework for privacy-preserving data-as-a-service mashups. In *IEEE 7th International Conference on Cloud Computing, Anchorage, USA, June 27 - July 2, 2014*, pages 498–505.

[13] Yigal Arens, Chin Y. Chee, Chun-Nan Hsu, and Craig A. Knoblock. Retrieving and integrating data from multiple information sources. *Int. J. Cooperative Inf. Syst.*, 2(2):127–158, 1993.

[14] David Aumueller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. Schema and ontology matching with coma++. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pages 906–908, New York, NY, USA, 2005. ACM.

[15] Oumayma Banouar and Said Raghay. Comparative study of the systems of semantic integration of information: A survey. In *12th IEEE/ACS International Conference of Computer Systems and Applications, AICCSA 2015, Marrakech, Morocco, November 17-20, 2015*, pages 1–8, 2015.

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

[17] Mahmoud Barhamgi and Djamal Benslimane. Composing data-providing web services. In *Proceedings of the VLDB,Lyon, France, August 24, 2009.*

[18] Mahmoud Barhamgi, Djamal Benslimane, and Brahim Medjahed. A query rewriting approach for web service composition. *IEEE Trans.Services Computing*, pages 206–222, 2010.

[19] Domenico Beneventano, Sonia Bergamaschi, Silvana Castano, Alberto Corni, R. Guidetti, G. Malvezzi, Michele Melchiori, and Maurizio Vincini. Information integration: The MOMIS project demonstration. In *Proc. of 26th Int. Conf. on Very Lare Data Bases*, pages 611–614, 2000.

[20] Jacob Berlin and Amihai Motro. Autoplex: Automated discovery of content for virtual databases. In Carlo Batini, Fausto Giunchiglia, Paolo Giorgini, and Massimo Mecella, editors, *Cooperative Information Systems*, pages 108–122, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[21] Geert Jan Bex, Sebastian Maneth, and Frank Neven. A formal model for an expressive fragment of xslt. *Information Systems*, 27(1):21–39, 2002.

[22] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, February 26 - March 1, 2002*, pages 431–440, 2002.

[23] Hugo Brunelière, Jordi Cabot, Grégoire Dupé, and Frédéric Madiot. Modisco: A model driven reverse engineering framework. *Information & Software Technology*, 56(8):1012–1032, 2014.

[24] Alexander Budanitsky and Graeme Hirst. Evaluating wordnet-based measures of lexical semantic relatedness. *Computational Linguistics*, 32(1):13–47, 2006.

[25] Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Paolo Naggar, and Fabio Vernacotola. Ibis: Semantic data integration at work. In Johann Eder and Michele Missikoff, editors, *Advanced Information Systems Engineering*, pages 79–94, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[26] M. Carey. Declarative data services: This is your data on soa. In *IEEE International Conference on Service-Oriented Computing and Applications (SOCA '07)*, pages 4–4, June 2007.

[27] Sudarshan S. Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey D. Ullman, and Jennifer Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *IPSJ*, pages 7–18, 1994.

[28] Víctor Cuevas-Vicenttín, Genoveva Vargas-Solar, and Christine Collet. Evaluating hybrid queries through service coordination in HYPATIA. In *15th International Conference on Extending Database Technology, EDBT '12, Berlin, Germany, March 27-30, 2012, Proceedings*, pages 602–605, 2012.

[29] Víctor Cuevas-Vicenttín, Genoveva Vargas-Solar, Christine Collet, Noha Ibrahim, and Christophe Bobineau. Coordinating services for accessing and processing data in dynamic environments. In *On the Move to Meaningful Internet Systems: OTM 2010 - Confederated International Conferences: CoopIS, IS, DOA and ODBASE, Hersonissos, Crete, Greece, October 25-29, 2010, Proceedings, Part I*, pages 309–325, 2010.

[30] Bolin Ding, Jeffrey Xu Yu, Shan Wang, Lu Qin, Xiao Zhang, and Xuemin Lin. Finding top-k min-cost connected trees in databases. In *Proceedings*

of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007, pages 836–845, 2007.

[31] Hong-Hai Do, Sergey Melnik, and Erhard Rahm. Comparison of schema matching evaluations. In Akmal B. Chaudhri, Mario Jeckle, Erhard Rahm, and Rainer Unland, editors, *Web, Web-Services, and Database Systems*, pages 221–237, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[32] Hong-Hai Do and Erhard Rahm. Coma: A system for flexible combination of schema matching approaches. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 610–621. VLDB Endowment, 2002.

[33] Hong Hai Do and Erhard Rahm. Matching large schemas: Approaches and evaluation. *Inf. Syst.*, 32(6):857–885, 2007.

[34] H. Elmeleegy, A. Ivan, R. Akkiraju, and R. Goodwin. Mashup advisor: A recommendation tool for mashup development. In *2008 IEEE International Conference on Web Services*, pages 337–344, Sept 2008.

[35] Hazem Elmeleegy, Anca Ivan, Rama Akkiraju, and Richard Goodwin. Mashup advisor: A recommendation tool for mashup development. In *2008 IEEE International Conference on Web Services (ICWS 2008), September 23-26, 2008, Beijing, China* [5], pages 337–344.

[36] Wenfei Fan and Floris Geerts. *Foundations of Data Quality Management*. Morgan & Claypool Publishers, 2012.

[37] G. Fandel and J. Spronk. *Multiple Criteria Decision Methods and Applications*. Springer Verlag, Berlin, 1985.

[38] Marc Friedman, Alon Levy, and Todd Millstein. Navigational plans for data integration. In *Proceedings of the 1999 International Conference on Intelligent Information Integration - Volume 23*, III'99, pages 72–78, Aachen, Germany, Germany, 1999. CEUR-WS.org.

[39] Samer Abdul Ghafour, Mahmoud Barhamgi, and Parisa Ghodous. On-demand data integration on the cloud. In *IEEE 7th International Conference on Cloud Computing, Anchorage, USA,June 27-July 2, 2014*, pages 924–927.

[40] Robert Mac Gregor. 13 - the evolving technology of classification-based knowledge representation systems. In JOHN F. SOWA, editor, *Principles of Semantic Networks*, The Morgan Kaufmann Series in Representation and Reasoning, pages 385 – 400. Morgan Kaufmann, 1991.

[41] Farshad Hakimpour and Andreas Geppert. Resolving semantic heterogeneity in schema integration. In *Proceedings of the International Conference on*

Formal Ontology in Information Systems - Volume 2001, FOIS '01, pages 297–308, New York, NY, USA, 2001. ACM.

[42] Alon Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, December 2001.

[43] Alon Y. Halevy, Anand Rajaraman, and Joann J. Ordille. Data integration: The teenage years. In *Proc. of the 32nd Int. Conf. on Very Large Data Bases*, pages 9–16, 2006.

[44] Yanbo Han, Guiling Wang, Guang Ji, and Peng Zhang. Situational data integration with data services and nested table. *Service Oriented Computing and Applications*, 7(2):129–150, 2013.

[45] Jon C. Ison, Matús Kalas, Inge Jonassen, Dan M. Bolser, Mahmut Uludag, Hamish McWilliam, James Malone, Rodrigo Lopez, Steve Pettifer, and Peter M. Rice. EDAM: an ontology of bioinformatics operations, types of data and identifiers, topics and formats. *Bioinformatics*, pages 1325–1332, 2013.

[46] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S. Sudarshan, Rushi Desai, and Hrishikesh Karambelkar. Bidirectional expansion for keyword search on graph databases. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, pages 505–516, 2005.

[47] Maurizio Lenzerini. Data integration: A theoretical perspective. In Lucian Popa, Serge Abiteboul, and Phokion G. Kolaitis, editors, *PODS*, pages 233–246. ACM, 2002.

[48] Alon Y. Levy. The information manifold approach to data integration. *IEEE Intelligent Systems*, 13:12–16, 1998.

[49] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Query-answering algorithms for information agents. In *IAAI, Portland, Oregon, August 4-8, 1996.*, pages 40–47.

[50] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 251–262, 1996.

[51] Wen-Syan Li, K. Selçuk Candan, Quoc Vu, and Divyakant Agrawal. Query relaxation by structure and semantics for retrieval of logical web documents. *IEEE Trans. Knowl. Data Eng.*, 14(4):768–791, 2002.

[52] Wen-Syan Li and Chris Clifton. Semint: A tool for identifying attribute correspondences in heterogeneous databases using neural networks, 2000.

[53] Li-Ren Liu, D. H. C. Du, and Hsi-Chuan Chen. An efficient parallel critical path algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(7):909–919, Jul 1994.

[54] Zohar Manna and Richard Waldinger. A deductive approach to program synthesis. *ACM Trans. Program. Lang. Syst.*, 2(1):90–121, January 1980.

[55] Sergey Melnik, Hector Garcia-molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching, 2002.

[56] George A Miller and Walter G Charles. Contextual correlates of semantic similarity. *Language Cognitive Processes*, 6(1):1–28, 1991.

[57] Steven Minton, Jaime G. Carbonell, Craig A. Knoblock, Daniel R. Kuokka, Oren Etzioni, and Yolanda Gil. Explanation-based learning:a problem solving perspective. *Artificial Intelligence*, 40(1):63 – 118, 1989.

[58] Mohamed Mussa, Samir Ouchani, Waseem Al Sammane, and Abdelwahab Hamou-Lhadj. A survey of model-driven testing techniques. In *Proceedings of the Ninth International Conference on Quality Software, QSIC 2009, Jeju, Korea, August 24-25, 2009*, pages 167–172, 2009.

[59] A. H. H. Ngu, M. P. Carlson, Q. Z. Sheng, and H. Paik. Semantic-based mashup of composite applications. *IEEE Transactions on Services Computing*, 3(1):2–15, Jan 2010.

[60] Xuan Thang Nguyen, Huu Tam Tran, Harun Baraki, and Kurt Geihs. FRASAD: A framework for model-driven iot application development. In *2nd IEEE World Forum on Internet of Things, WF-IoT 2015, Milan, Italy, December 14-16, 2015*, pages 387–392, 2015.

[61] Andrew Nightingale, Ricardo Antunes, Emanuele Alpi, Borisas Bursteinas, Leonardo Gonzales, Wudong Liu, Jie Luo, Guoying Qi, Edd Turner, and Maria Jesus Martin. The proteins API: accessing key integrated protein and genome information. *Nucleic Acids Research*, (Webserver-Issue):W539–W544, 2017.

[62] Neelu Nihalani, Sanjay Silakari, and Mahesh Motwani. Integration of artificial intelligence and database management system: An inventive approach for intelligent databases. In *Proceedings of the 2009 First International Conference on Computational Intelligence, Communication Systems and Networks*, CICSYN '09, pages 35–40, Washington, DC, USA, 2009. IEEE Computer Society.

[63] Siddharth Patwardhan and Ted Pedersen. Using wordnet-based context vectors to estimate the semantic relatedness of concepts. In *Proceedings of the*

EACL 2006 Workshop Making Sense of Sense-Bringing Computational Linguistics and Psycholinguistics Together, volume 1501, pages 1–8, 2006.

[64] Ted Pedersen, Siddharth Patwardhan, and Jason Michelizzi. Wordnet: : Similarity - measuring the relatedness of concepts. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, pages 1024–1025, 2004.

[65] Nicoleta Preda, Gjergji Kasneci, Fabian M. Suchanek, Thomas Neumann, Wenjun Yuan, and Gerhard Weikum. Active knowledge: dynamically enriching RDF knowledge bases by web services. In *Proc.of the ACM SIGMOD Int. Conf. on Management of Data*, pages 399–410, 2010.

[66] Nicoleta Preda, Fabian M. Suchanek, Gjergji Kasneci, Thomas Neumann, Maya Ramanath, and Gerhard Weikum. ANGIE: active knowledge for interactive exploration. *PVLDB*, 2(2):1570–1573, 2009.

[67] D. Pukhkaiev, T. Kot, L. Globa, and A. Schill. A novel sla-aware approach for web service composition. In *Eurocon 2013*, pages 327–334, July 2013.

[68] Christoph Quix. Managing Data lakes in big data era. In *Proc. 5th Int. Conf. on Cyber Technology in Automation, Control and Intelligent Systems*, pages 820–824, 2015.

[69] Gabriele Reich and Peter Widmayer. Beyond steiner's problem: A vlsi oriented generalization. In Manfred Nagl, editor, *WG*, volume 411 of *Lecture Notes in Computer Science*, pages 196–210. Springer, 1989.

[70] Gabriele Reich and Peter Widmayer. Beyond steiner's problem: A vlsi oriented generalization. In Manfred Nagl, editor, *Graph-Theoretic Concepts in Computer Science*, pages 196–210, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.

[71] Herbert Rubenstein and John B. Goodenough. Contextual correlates of synonymy. *Commun. ACM*, 8(10):627–633, October 1965.

[72] Manivasakan Sabesan and Tore Risch. Adaptive parallelization of queries over dependent web service calls. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pages 1725–1732, 2009.

[73] Pushpak Sarkar. *Introduction to DaaS*, pages 368–. Wiley-IEEE Press, 2015.

[74] Ángel Mora Segura, Jesús Sánchez Cuadrado, and Juan de Lara. Odaas: Towards the model-driven engineering of open data applications as data services. In *EDOC Workshops*, pages 335–339. IEEE Computer Society, 2014.

[75] Jun'ichi Tatemura, Songting Chen, Fenglin Liao, Oliver Po, K. Selçuk Candan, and Divyakant Agrawal. UQBE: uncertain query by example for web service mashup. In *SIGMOD, Canada, June*, pages 1275–1280, 2008.

[76] Junichi Tatemura, Arsany Sawires, Oliver Po, Songting Chen, K. Selcuk Candan, Diviyakant Agrawal, and Maria Goveas. Mashup feeds: Continuous queries over web services. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 1128–1130, New York, NY, USA, 2007. ACM.

[77] H. L. Truong and S. Dustdar. On analyzing and specifying concerns for data as a service. In *2009 IEEE Asia-Pacific Services Computing Conference (APSCC)*, pages 87–94, Dec 2009.

[78] Hong Linh Truong and Schahram Dustdar. On evaluating and publishing data concerns for data as a service. In *5th IEEE Asia-Pacific Services Computing Conference, APSCC 2010, 6-10 December 2010, Hangzhou, China, Proceedings*, pages 363–370, 2010.

[79] Rattapoom Tuchinda, Craig A. Knoblock, and Pedro A. Szekely. Building mashups by demonstration. *TWEB*, 5(3):16:1–16:45, 2011.

[80] Rattapoom Tuchinda, Pedro Szekely, and Craig A. Knoblock. Building data integration queries by demonstration. In *Proceedings of the 12th International Conference on Intelligent User Interfaces*, IUI '07, pages 170–179, New York, NY, USA, 2007. ACM.

[81] Roman Vaculín, Huajun Chen, Roman Neruda, and Katia P. Sycara. Modeling and discovery of data providing services. In *2008 IEEE International Conference on Web Services (ICWS 2008), September 23-26, 2008, Beijing, China* [5], pages 54–61.

[82] Quang Hieu Vu, Tran Vu Pham, Hong Linh Truong, Schahram Dustdar, and Rasool Asal. DEMODS: A description model for data-as-a-service. In *AINA*, pages 605–612. IEEE Computer Society, 2012.

[83] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z. Sheng. Quality driven web services composition. In *Proc. of the 12th International Conference on WWW*, pages 411–421, New York,USA, 2003.

[84] Patrick Ziegler and Klaus R. Dittrich. Three decades of data integration - all problems solved? In *Building the Information Society, IFIP 18th World Computer Cong.*, pages 3–12, 2004.

RÉSUMÉ

De nos jours, d'énormes volumes de données sont créés en continu et les utilisateurs s'attendent à ce que ceux-ci soient collectés, stockés et traités quasiment en temps réel. Ainsi, les lacs de données sont devenus une solution attractive par rapport aux entrepôts de données classiques coûteux et fastidieux (nécessitant une démarche ETL), pour les entreprises qui souhaitent stocker leurs données. Malgré leurs volumes, les données stockées dans les lacs de données des entreprises sont souvent incomplètes voire non mises à jour vis-à-vis des besoins (requêtes) des utilisateurs. Les sources de données locales ont donc besoin d'être enrichies. Par ailleurs, la diversité et l'expansion du nombre de sources d'information disponibles sur le web a rendu possible l'extraction des données en temps réel. Ainsi, afin de permettre d'accéder et de récupérer l'information de manière simple et interopérable, les sources de données sont de plus en plus intégrées dans les services Web. Il s'agit plus précisément des services de données, y compris les services DaaS du Cloud Computing. L'enrichissement manuel des sources locales implique plusieurs tâches fastidieuses telles que l'identification des services pertinents, l'extraction et l'intégration de données hétérogènes, la définition des mappings service-source, etc. Dans un tel contexte, nous proposons une nouvelle approche d'intégration de données centrée utilisateur. Le but principal est d'enrichir les sources de données locales avec des données extraites à partir du web via les services de données. Cela permettrait de satisfaire les requêtes des utilisateurs tout en respectant leurs préférences en terme de coût d'exécution et de temps de réponse et en garantissant la qualité des résultats obtenus.

MOTS CLÉS

Intégration de données centrée utilisateur, Lacs de Services, Enrichissment de schéma, Composition des services, Services de Données, Qualité de données, Qualité des Services, Préférences Utilisateur, Nuage de données, Données en tant que Service, Ingénierie Dirigée par les Modèles, Annotation Sémantique.

ABSTRACT

In the Big Data era, companies are moving away from traditional data-warehouse solutions whereby expensive and time-consuming ETL (Extract, Transform, Load) processes are used, towards data lakes in order to manage their increasingly growing data. Yet the stored knowledge in companies' databases, even though in the constructed data lakes, can never be complete and up-to-date, because of the continuous production of data. Local data sources often need to be augmented and enriched with information coming from external data sources. Unfortunately, the data enrichment process is one of the manual labors undertaken by experts who enrich data by adding information based on their expertise or select relevant data sources to complete missing information. Such work can be tedious, expensive and time-consuming, making it very promising for automation. We present in this work an active user-centric data integration approach to automatically enrich local data sources, in which the missing information is leveraged on the fly from web sources using data services. Accordingly, our approach enables users to query for information about concepts that are not defined in the data source schema. In doing so, we take into consideration a set of user preferences such as the cost threshold and the response time necessary to compute the desired answers, while ensuring a good quality of the obtained results.

KEYWORDS

User-Centric Data Integration, Data Provisioning Service Lakes, Schema Enriching, Service Composition, Data Services, Service views, Data Quality, Quality of Service (QoS), User Preferences, Cloud Computing, Data as a Service (DaaS), Model Driven Engineering, Semantic Annotation.