République Algérienne Démocratique et Populaire

Université Abou Bakr Belkaid– Tlemcen

Faculté des Sciences

Département d' Informatique

MÉMOIRE

Pour l'obtention du diplôme de

MASTER EN INFORMATIQUE

Réalisé par

BOUHASSOUN Jamel & BOUHASSOUN Leila

Interrogation de sources de données médicales à base de Services Web DaaS

Présenté le 29 septembre 2011 devant la commission composé de MM.

Président :	- Abderrahim M.A.
Examinateurs :	- Benamar A.
	- Chouiti S.
	- El yebdri Z.
	- Halfaoui A.
	- Benmansour F.
Encadreurs :	- Midouni S.D.
	- Settouti S.L.

Année Universitaire : 2010 - 2011

Résumé

La composition des Services Web permet de répondre aux besoins d'un utilisateur qui ne peuvent être satisfaits par un seul Service Web. Dans ce travail, nous nous intéressons à automatiser la composition des Services Web DaaS (Data-as-a-Service Web Services) en tenant compte de la relation sémantique entre les entrées/sorties d'un service.

Pour atteindre notre objectif, nous modélisons nos Services Web DaaS par des vues RDF/RDFS sur une ontologie de médiation. En plus, nous proposons une approche de réécriture de requête qui permet la sélection, l'invocation et la composition des Services Web DaaS afin de répondre à une requête d'utilisateur.

Mots-clés: Services Web, composition des Services Web, Services Web DaaS, RDF/RDFS, ontologie de médiation, réécriture de requête.

Abstract

Web Services composition allows to answer the needs of a user that can not be satisfied by a single Web Service. In this work, we focus to automate DaaS (Data-as-a-Service) Web Services composition taking into account the semantic relationship between the inputs/outputs of a service.

To achieve our goal, we model our DaaS Web Services by RDF/RDFS views over a mediated ontology. In addition, we propose an approach to rewrite queries that allows the selection, invocation and composition of DaaS Web Services to answer a user query.

Keywords: Web Services, Web Services composition, DaaS Web Services, RDF/RDFS, mediated ontology, query rewriting.

Remerciements

Nous tenons tout d'abord à remercier DIEU le tout puissant pour nous avoir donné la force de réaliser ce travail, et aussi nos parents pour leurs soutiens et encouragements durant nos années d'études.

Nous remercions profondément notre encadreur Mr. MIDOUNI Sid Ahmed Djallal pour son aide, ses encouragements et ses critiques constructives qui nous ont beaucoup aidé à apprécier ce travail.

Un remerciement spécial est adressé à notre co-encadreur Mr. SETTOUTI Lotfi Sofiane pour ses conseils et pour les nombreux éclaircissements qu'il a bien voulu nous apporter durant tout ce projet.

Nous remercions également les membres du jury qui ont accepté de participer à la discussion de notre travail.

Nous exprimons aussi toute notre gratitude aux enseignants du département d'informatique ainsi que tous nos collègues et nos amis et à tous ce qui ont contribué de près ou de loin à la réalisation de notre mémoire.

A nos parents

A nos familles

A nos amis(es)

Table des matières

Table des figures

Liste des tableaux

Introduction Générale

Les besoins d'accéder de façon uniforme à des sources de données multiples sont chaque jour de plus en plus fort, particulièrement, dans les systèmes décisionnels qui ont besoin d'une analyse compréhensive des données. Les systèmes d'intégration de données sont apparus pour répondre à ces besoins. Un système d'intégration de données est un système d'information qui intègre des données de sources différentes et fournit à l'utilisateur une vision d'une base de données unique. En effet, les sources de données sont le plus souvent *réparties*, *autonomes* et *hétérogènes*.

Les systèmes de médiation sont des exemples de systèmes d'intégration de données qui fournissent un accès à des données extraites à partir de diverses sources de données et permettent de les intégrer pour répondre aux requêtes des utilisateurs. Ils permettent aussi de découvrir les sources pertinentes étant donné une requête, accéder à ces sources et enfin combiner automatiquement les réponses partielles obtenues de plusieurs sources de façon à délivrer une réponse globale. Un système de médiation est spécifique à un domaine d'application donné. Il comprend un *schéma global* ou *ontologie* dont le rôle est central. C'est un modèle du domaine d'application du système. Il fournit un vocabulaire structuré servant de support à l'expression des requêtes des utilisateurs et la description des contenus des sources par un ensemble de vues. Par ailleurs, il établit une connexion entre les différentes sources accessibles.

L'approche de médiation présente l'intérêt de pouvoir construire un système d'interrogation de sources de données à base de Services Web sans toucher aux données qui restent stockées dans leurs sources d'origine. En effet, un Service Web est une entité logicielle permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués. Etant donné qu'il y a plusieurs Services

Web publiés sur Internet, beaucoup d'entre eux ne peuvent seuls satisfaire les différents besoins complexes d'un utilisateur. De plus, il arrive qu'un besoin quelconque en termes de fonctionnalité ne puisse pas être satisfait par un seul Service Web, mais, par la combinaison d'un ensemble de Services Web. Il sera donc nécessaire dans ce cas d'ordonner plusieurs Services Web basiques en un Service Web plus complexe. Pour répondre à une requête d'utilisateur, un médiateur doit reformuler la requête initiale en un plan de requêtes directement exécutables sur les sources de données disponibles. Cette reformulation est obtenue par *la réécriture de requête en termes de vues*. Les Services Web composites ainsi obtenus peuvent entrer à leur tour dans d'autres compositions pour fournir des Services Web plus complexes à forte valeur ajoutée.

Problématique

Le problème majeur, auquel nous nous intéresserons dans notre travail est le problème de réécriture de requête en termes de vues. Etant donné un ensemble de Services Web et une requête d'utilisateur, comment trouver les combinaisons de services qui satisfont au mieux la requête ?. C'est à la fois le problème de localisation, identification et composition automatique de Services Web dont l'enjeu est de créer des Services Web complexes qui résolvent le problème à partir de Services Web plus simples avec l'étude de l'apport des descriptions sémantiques qui leurs sont associées. Notre objectif principal est de proposer un système de médiation qui concerne le domaine médical à base de Services Web DaaS. Ce système permet de répondre aux requêtes des utilisateurs qui sont souvent complexes et sophistiqués par composition de Services Web DaaS.

Contribution

Dans le cadre de ce travail, nous décrivons un scénario de motivation nécessitant la composition de Services Web DaaS et nous proposons un ensemble de services DaaS qui peuvent être utilisés par le système de médiation pour l'interrogation des sources de données structurées (e.g. BDDR, XML). Ensuite, nous présentons une approche de réécriture de requête pour la composition automatique de Services Web DaaS. Nous proposons également notre ontologie de médiation qui concerne le domaine médical permettant de fournir le vocabulaire nécessaire pour l'expression des requêtes des utilisateurs ainsi que la description des contenus des sources par un ensemble de vues RDF/RDFS. Enfin, nous

présentons notre algorithme de réécriture de requête qui permet la sélection, l'invocation et la composition de Services Web DaaS afin de répondre aux requêtes d'utilisateur.

Ce mémoire est organisé en trois chapitres :

Chapitre 1 : Notions Préliminaires. Dans ce chapitre, nous présentons les concepts de base concernant les Services Web, les Services Web DaaS et les Services Web sémantiques. Nous donnons la définition de l'ontologie et nous parlons de son rôle très important pour le Web sémantique, puis nous citons les langages de description RDF/RDFS et le langage d'interrogation SPARQL.

Chapitre 2 : Etat de l'art sur la composition des Services Web. Dans ce chapitre, nous commençons tout d'abord à introduire la notion de composition des Services Web avec leurs approches existantes. Ensuite, nous présentons les travaux relatifs au problème de réécriture de requêtes en termes de vues et enfin, nous discutons notre travail par rapport aux autres travaux.

Chapitre 3 : Conception de notre système de médiation. Ce chapitre est dédié à la conception de notre système de médiation. Nous commençons tout d'abord à décrire un scénario de motivation pour la composition des services DaaS. Ensuite, nous présentons notre approche pour la composition, notre modèle de requête proposé et notre ontologie de médiation ainsi que la description des services DaaS proposés. Enfin, nous présentons notre algorithme de réécriture de requête permettant la composition des Services Web DaaS afin de répondre à la requête d'utilisateur.

Enfin, nous terminons ce mémoire par une conclusion générale dans laquelle nous discutons notre travail et nous proposons quelques perspectives de recherche que nous souhaitons les réaliser dans le futur.

Chapitre 1

Notions Préliminaires

Sommaire

1.1 Introduction

Dans ce chapitre, nous présentons quelques concepts de base qui nous semblent indispensables. Dans la section 1.2, nous donnons quelques définitions du concept "Service Web" et nous présentons son modèle dans la section 1.3. De la même façon, nous introduisons les concepts "Service Web DaaS" et "Services Web sémantiques" respectivement dans les sections 1.4 et 1.5. Ensuite, nous définissons le concept "ontologie" (section 1.6), les langages de description RDF/RDFS (section 1.7) et enfin, le langage d'interrogation SPARQL (section 1.8).

1.2 Qu'est ce qu'un Service Web ?

Le Web est de plus en plus le support privilégié des applications. Les Services Web constituent le développement ultime dans ce domaine. Un Service Web est souvent vu comme une application accessible à d'autres applications sur le Web. Cependant, il existe plusieurs définitions pour le Service Web, nous citerons quelques unes :

Définition 1 : Le consortium W3C [1] définit un Service Web comme étant :« **une application, ou un composant logiciel qui vérifie les propriétés suivantes** » [23] :

- Il est identifié par un URI [2] ;
- Ses interfaces et ses liens peuvent être décrits en XML ;
- Sa définition peut être découverte par d'autres Services Web ;
- Il peut interagir directement avec d'autres Services Web à travers le langage XML en utilisant des protocoles Internet standards.

Définition 2 : « Un Service Web est une application accessible à partir du Web. Il utilise les protocoles Internet pour communiquer et un langage standard pour décrire son interface » [12].

Définition 3 : « Les Services Web sont la nouvelle vague des applications Web. Ce sont des applications modulaires, auto-contenues et auto-descriptives qui peuvent être publiées, localisées et invoquées depuis le Web. Les Services Web effectuent des actions allant de simples requêtes à des processus métiers complexes. Une fois qu'un Service Web est déployé, d'autres applications (y compris des Services Web) peuvent le découvrir et l'invoquer » [14].

1. World Wide Web Consortium
2. Uniform Resource Identifier

1.3 Modèle de base des Services Web

Le modèle des Services Web repose sur une Architecture Orientée Service (SOA) [3]. Celle-ci fait intervenir trois catégories d'acteurs : **le fournisseur de service**, **l'annuaire des services** et **le client**. Un fournisseur de service fournit un module logiciel accessible sur le réseau et définit une description de service pour le Service Web. Ensuite, il le publie dans l'annuaire des services de telle sorte que le client peut le trouver. L'annuaire des services correspond à un registre de descriptions des Services Web offrant des facilités de publication des Services Web à l'intention des fournisseurs, ainsi que des facilités de recherche des Services Web à l'intention des clients. La description de service contient des informations telles que l'entrée du service, sa sortie et l'adresse où le service est situé. Le client interroge l'annuaire pour un certain type de service et récupère sa description. Ensuite, il utilise les informations dans la description du service pour se lier avec le fournisseur et d'invoquer le Service Web [6]. Les intéractions de base entre ces trois acteurs montrées dans la figure 1.1 incluent les opérations de publication, de recherche et de liens d'opérations. Nous citons, notamment les standards émergents suivants :

- **SOAP** [4] **:** Définit un protocole de transmission de messages basé sur XML. Il permet l'échange d'informations à distance en utilisant le formalisme XML pour à la fois, définir les messages envoyés entre les applications et représenter les données échangées.
- **WSDL** [5] **:** Un standard fondé sur XML qui permet la description des Services Web.
- **UDDI** [6] **:** Fournit l'infrastructure de base pour la publication et la recherche des Services Web.

3. Service Oriented Architecture
4. Simple Object Access Protocol
5. Web Service Description Language
6. Universal Description Discovery and Integration

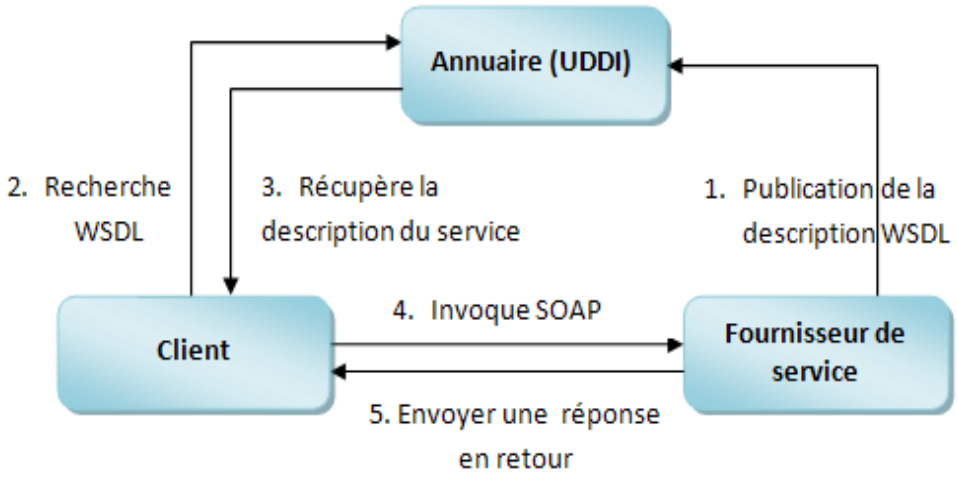

FIGURE 1.1 – Modèle du Service Web

1.4 Les Services Web DaaS (Data as a Service)

Au cours de la dernière décennie, les Services Web ont été largement perçus comme un moyen standardisé pour l'intégration d'applications sur le Web. Ils reposent sur une architecture orientée service permettant ainsi aux applications de différents fournisseurs d'être encapsulées comme des services, puis publiées, localisées, invoquées, composées et coordonnées de manière à couplage faible. Récemment, les Services Web ont commencé à être un support populaire pour la publication et le partage des données sur le Web.

Les entreprises modernes évoluent vers une architecture orientée service de partage de données sur le Web en mettant leurs bases de données derrière les Services Web, fournissant ainsi la méthode interopérable d'interagir avec leurs données. En outre, les données qui ne sont pas stockées dans des bases de données traditionnelles sont également mises à disposition via des Services Web. Nous appelons ce type de Services Web en tant que `Services Web DaaS`.

« Un Service DaaS fournit une vue simplifiée, intégrée en temps réel , une information de haute qualité sur une entité commerciale spécifique comme un client ou un produit. Il peut être fourni par le middleware ou emballé comme un composant logiciel individuel. Les informations qu'il fournit proviennent d'un ensemble diversifié de ressources d'informations » [11].

Une autre définition de Composite Software : « **Les Services Web DaaS sont une forme de Services Web optimisée pour les demandes d'intégration de données en temps réel. Ils visualisent les données pour découpler les emplacements physiques et logiques et donc, d'éviter la réplication des données inutiles. Les Services Web DaaS résument des structures de données complexes et de la syntaxe. Ils fédèrent des données disparates en composites utiles et supportent l'intégration de données à travers les applications de l'architecture orientée service** »[7].

Les Services Web DaaS permettent l'accès aux sources de données des organisations. L'invocation d'un service DaaS résulte dans l'exécution d'une requête sur le schéma de la source de données. Lorsqu'un tel service est exécuté, il accepte d'un utilisateur une donnée d'entrée d'un format spécifié et il lui retourne des informations comme une sortie. Les Services Web DaaS sont maintenant utilisés dans de nombreux domaines d'application comme un moyen standard pour la publication et le partage des données. Exemples de domaines d'application comprennent, entre autres, le partage des données scientifiques (par exemple, la bioinformatique, traitement et partage de données géospatiales , etc), le partage des données médicales (eHealth), les entreprises d'intégration de données, le partage des données entre les organismes gouvernementaux (eGovernment), etc).

1.5 Les Services Web sémantiques

Les Services Web sémantiques se situent à la convergence de deux domaines de recherche importants qui concernent les technologies d' Internet : le Web sémantique et les Services Web. Cette tâche de convergence est accomplie en rendant les Services Web auto-exploitables par machines, et de réaliser l'interopérabilité entre les applications via le Web en vue de rendre le Web plus dynamique.

En effet, le Web sémantique a été inventé en 1989 par Tim Berners-Lee. Ce dernier a proclamé que le Web sémantique est la prochaine évolution du Web, c'est-à-dire que l'on

7. http ://compositesoftware.com/solutions/soa.shtml

va arriver à un Web intelligent où les informations sont stockées de façon compréhensible par les machines afin d'apporter à l'utilisateur ce qui cherche vraiment. Aujourd'hui, les humains sont les seuls qui ont la capacité de comprendre ce que nous trouvons et de décider en quoi cela se rapporte à ce que nous voulons chercher vraiment. Par quels moyens ? Ce sont les moteurs de recherche qui nous aident, mais ils sont capables de répondre seulement aux deux questions : Quelles sont les pages contenant un terme ? et Quelles sont les pages les plus populaires à un sujet ?.

« **Le Web sémantique est une extension du Web actuel dans lequel l'information est munie d'une signification bien définie permettant aux machines et aux personnes de travailler en coopération** » [19]. Le Web sémantique reste entièrement fondé sur le Web classique et ne le remet pas en cause. Cela reste un moyen de publier et consulter des documents, mais les documents traités par le Web sémantique contiennent non pas des textes en langage naturel mais des informations formalisées pouvant être traitées automatiquement par des agents logiciels. Un des objectifs du Web sémantique est d'affiner la recherche sur Internet. Ceci est réalisé par l'annotation du contenu du Web en ajoutant aux informations existantes une couche de métadonnées pour améliorer la compréhension pour la machine. Pour le faire, on utilise le vocabulaire conceptuel fourni par une *ontologie*.

Un Service Web sémantique est un Service Web décrit en utilisant des annotations sémantiques dans un langage du Web sémantique bien défini, qui permettent au Service Web d'avoir une interface compréhensible par les humains et les machines. Ces Services Web sémantiques s'appuient en général sur les langages du Web sémantique pour décrire leurs fonctionnalités et les données qu'ils échangent. Les motivations pour développer, ou tendre vers les Services Web sémantiques sont évidemment de faciliter les phases automatiques de découverte, sélection et composition de Services Web. En effet, si leur sémantique est connue, alors chercher et composer des Services Web pourra être fait automatiquement en donnant la sémantique cible.

1.6 Ontologies pour le Web sémantique

L'ontologie joue un rôle très important pour le Web sémantique parce qu'elle représente la sémantique des documents en permettant leur exploitation par les applications et les agents intelligents. Elle est très utile pour structurer et définir la signification des termes de métadonnées actuellement collectées et normalisées. Donc, à l'aide des ontologies, les applications sur le Web de demain pourrons devenir intelligentes, au sens où elles pourront opérer plus précisément au niveau conceptuel humain.

Une des définitions de l'ontologie qui fait autorité est celle de Gruber [17] : « **Une ontologie est une spécification formelle et explicite d'une conceptualisation partagée** ». La ***conceptualisation*** représente la collection des objets, de concepts et des autres entités qui sont supposés exister dans un certain domaine d'intérêt, et les relations qui les relient. Une conceptualisation est une vue abstraite, simplifiée du monde que l'on veut représenter. ***Explicite*** signifie que le type des concepts utilisés et les contraintes sur leur utilisation sont explicitement définis. ***Formelle*** se réfère au fait que l'ontologie doit être compréhensible par les machines. ***Partagée*** reflète la notion de connaissance consensuelle décrite par l'ontologie, c'est-à-dire qu'elle n'est pas restreinte au point de vue de certains individus seulement, mais reflète un point de vue plus général, partagé et accepté par un groupe.

Une ontologie est une structure de donnée opérationnelle qui rend compte des concepts d'un domaine et de leurs relations. Le développement des ontologies croissant en Intelligence Artificielle vient de leur intérêt pour associer du sens à des ressources textuelles, pour localiser et gérer des connaissances dans diverses applications. Son but est donc de définir quelles primitives avec leur sémantique associée sont nécessaires pour la représentation des connaissances dans un contexte donné. Elle est représentée par un graphe orienté qui contient :

- Les nœuds (concepts) représentant le vocabulaire d'un domaine particulier.
- Les arcs représentant les relations (ou rôles) nommées entre les concepts.

La figure suivante montre un exemple d'ontologie pour les images pneumologiques :

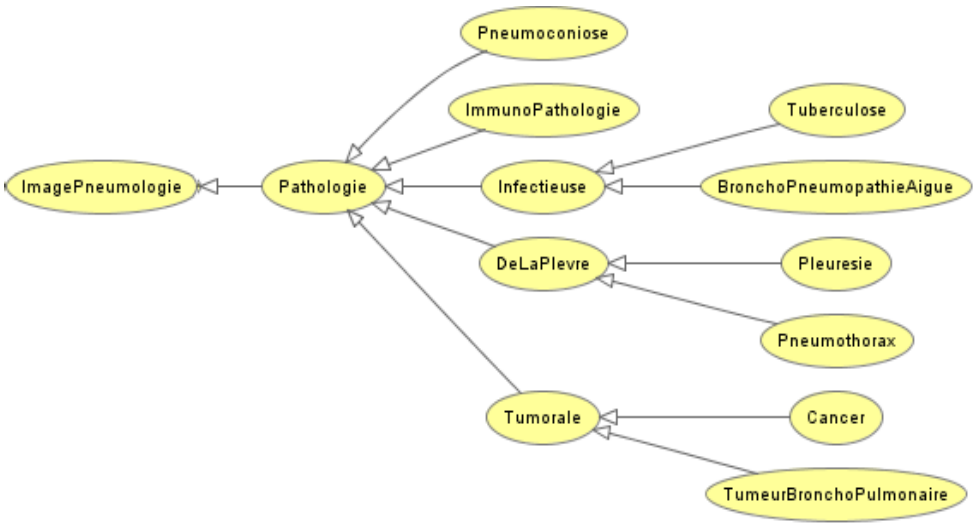

FIGURE 1.2 – Exemple d'une partie d'ontologie des images médicales (images pneumologiques) [1]

A partir de cette structure, la sémantique de chaque mot est déduite par les relations que ce mot possède dans l'ontologie, ce qui permet de restreindre les interprétations possibles. Cependant pour être exploitable par une machine, une ontologie doit respecter certaines règles :

- Etre définie par une syntaxe formelle et une sémantique non ambigüe.
- Permettre la déduction de nouvelles connaissances qui sont présentées implicitement.

De plus, pour assurer sa pérennité, une ontologie doit posséder un niveau d'abstraction permettant son extension [15].

1.7 RDF/RDFS

1. « **RDF (Resource Description Framework) est un modèle standard pour l'échange de données sur le Web. Il permet de décrire de façon formelle les ressources Web et leurs métadonnées** » [21]. En utilisant RDF, les ressources sont décrites par un ensemble de déclarations RDF sous forme de triplets `<Sujet,Propriété,Objet>` où :

– *Sujet :* représente la ressource à décrire. Il est identifié par une URI.
– *Propriété ou prédicat :* il s'agit d'une propriété utilisée pour caractériser et décrire une ressource. Une propriété est une liaison étiquetée et orientée du sujet vers l'objet. Elle est identifiée par une URI.
– *Objet :* la valeur de la propriété pouvant être un littéral ou bien une autre ressource (identifiée par une URI).

Cet ensemble de triplets RDF peut être représenté par un graphe orienté étiqueté (Figure 1.3) où les éléments apparaissant comme sujet ou objet sont des nœuds et les propriétés sont des arcs.

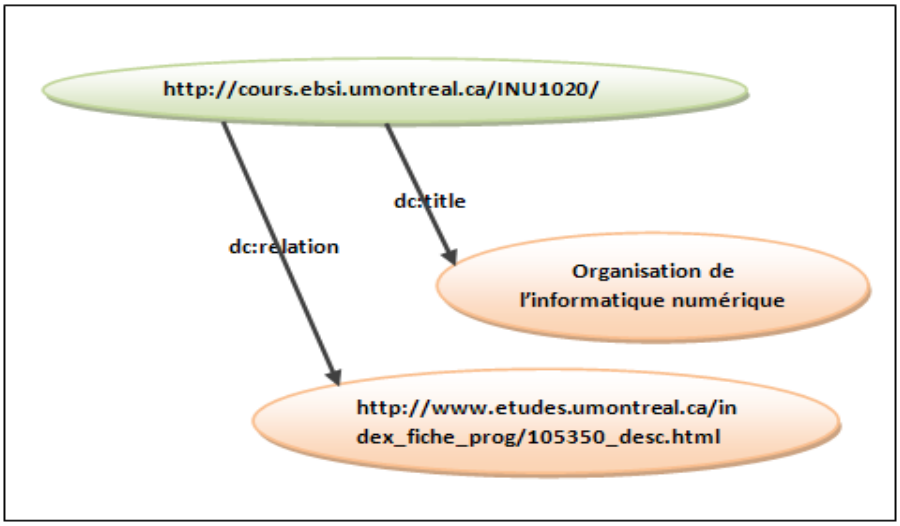

FIGURE 1.3 – Exemple de graphe RDF.

Les nœuds peuvent se présenter sous la forme :

– **Valeur littérale** : Ex : Michel Tremblay.
– **URI**.
– **Vide** : permet de désigner une ressource par ses propriétés sans expliciter cette ressource.

Les arcs peuvent se présenter sous la forme :

– **URI** :
  – **Forme longue** : http ://purl.org/dc/elements/1.1/title
  – **Forme raccourcie** : faisant appel à un préfixe dc :title

A noter : un préfixe c'est un raccourci de l'URI complet de la ressource. Ecrire dc :title équivaut à écrire l'URI de la propriété dans sa forme longue.

2. **RDFS (Resource Description Framework Schema)**

   Il est nécessaire, pour donner une sémantique ou un sens aux informations stockées sous forme de triplets RDF, de se donner un vocabulaire (classes et propriétés).
   « RDFS est un langage extensible de représentation des connaissances qui permet de décrire précisément par vocabulaires les ressources d'un domaine donné et les relations entre elles. RDFS fournit des éléments de base pour la définition d'ontologies ou vocabulaires destinés à structurer des ressources RDF » [20].

   Il permet d'organiser les classes en une hiérarchie de classes en utilisant les relations de subsomption entre classes "`subClassOf`" et d'organiser les propriétés en une hiérarchie de propriétés en utilisant des relations de subsomption entre propriétés "`subPropertyOf`". De plus, RDFS offre le moyen de spécifier le typage des propriétés en indiquant leur "`domaine`" et leur "`co-domaine`" `(range)`.

1.8 SPARQL

« SPARQL est un langage de requêtes pour l'interrogation de métadonnées et l'extraction des données sous forme d'un graphe RDF ou plus exactement un langage d'interrogation de triplets RDF » [22].

SPARQL est adapté à la structure spécifique des graphes RDF et s'appuie sur les triplets qui les constituent . Il définit la syntaxe et la sémantique nécessaire à l'expression de requêtes sur une base de données de type RDF. Il est différent du classique SQL (langage de requête qui est adapté aux bases de données relationnelles) mais s'en inspire clairement dans sa syntaxe et ses fonctionnalités. Il a aussi quelques traits de ressemblances mineures avec Prolog. SPARQL permet d'exprimer des requêtes interrogatives ou constructives :

- Une requête SELECT de type interrogative permet d'extraire du graphe RDF un sous-graphe correspondant à un ensemble de ressources vérifiant les conditions dé-

finies dans une clause WHERE.

– Une requête CONSTRUCT de type constructive, engendre un nouveau graphe qui complète le graphe interrogé.

La structure d'une requête SPARQL est très similaire à celle employée dans le langage SQL :

SELECT $?v_1 ?v_2 \ldots ?v_n$

FROM <description.rdf>

WHERE {

(sujet1 |vi) (predicat1 | vj) (objet1 | vk) ... (sujetx |va) (predicaty | vb) (objetz | vc)

}

1.9 Conclusion

Les Services Web représentent aujourd'hui la technologie la plus adaptée pour assurer le développement des systèmes distribués sur Internet. Un des concepts intéressants qu'offre la technologie de Service Web et qui suscite beaucoup d'intérêt est la possibilité de créer un nouveau Service Web à valeur ajoutée par composition de plusieurs Services Web existants.

Chapitre 2

Etat de l'art sur la composition des Services Web

Sommaire

2.1 Introduction

Ce chapitre est essentiellement consacré à l'état de l'art relatif à la composition des Services Web. Nous commençons tout d'abord par introduire le besoin de composition des Services Web dans la section 2.2 et nous présentons les approches existantes pour la composition des Services Web dans la section 2.3. Ensuite, nous introduisons le problème de réécriture de requêtes en termes de vues pour lequel, nous présentons trois algorithmes (section 2.4). Enfin, nous discutons notre travail par rapport aux autres travaux.

2.2 Composition des Services Web

Ces dernières années, la recherche dans le domaine des Services Web a été très développée. Une grande partie de cette recherche a été consacrée à la composition de Services Web. Réellement, il n'est pas toujours facile de trouver des Services Web qui s'apparient avec les requêtes des utilisateurs. Par conséquent, la composition des services satisfaisant la requête est un besoin grandissant de nos jours. La composition de services est une tâche complexe. Cette complexité est due principalement aux raisons suivantes [4] :

- Le nombre de Services Web disponibles sur le Web est potentiellement très important et en constante augmentation ;
- Les Services Web peuvent être créés et modifiés, le système de composition doit donc détecter et gérer ces changements ;
- Les Services Web sont créés par des organisations différentes qui n'ont pas forcément les mêmes modèles de concepts pour décrire ces services.

La composition des Services Web devient de plus en plus incontournable dans un environnement ouvert et dynamique comme Internet. Il est évident qu'une composition à la demande sera très préférable, en plus, il sera judicieux de prendre en charge la sémantique durant la composition des Services Web afin de minimiser les fausses réponses et d'améliorer la qualité globale des résultats.

La composition des Services Web est le processus de construction de nouveaux Services Web à valeur ajoutée à partir de deux ou plusieurs Services Web déjà présents et publiés sur le Web. Un Service Web est dit composé ou composite lorsque son exécution implique des interactions avec d'autres Services Web et des changements des messages entre eux afin de faire appel à leurs fonctionnalités. La composition de Services Web spécifie quels services ont besoin d'être invoqués, dans quel ordre et comment gérer les conditions d'interactions [5].

2.3 Approches existantes

Il existe principalement deux grandes approches de composition des Services Web : *l'approche statique* et *l'approche dynamique*. Nous présentons brièvement chacune de ces approches dans ce qui suit.

2.3.1 Composition statique des Services Web

Une composition est statique quand tous les Services Web qui font partie de la composition sont connus, ainsi que leur ordre d'exécution. Dans cette composition, les services sont connus avant leur exécution, c'est-à-dire, au moment où la composition est faite. De plus, nous connaissons également l'ordre d'exécution ainsi que la localisation et la description des Services Web. Dans la composition statique, tout est complètement spécifié et en particulier tous les Services Web sont connus. [7]

De plus, la création des processus métiers se fait durant le développement du système et reste statique pendant son utilisation, or l'environnement des Services Web est dynamique. Pour ces raisons, les chercheurs fournissent beaucoup d'efforts pour la création des processus métiers et le rendre plus dynamique en restreignant, autant que possible, l'intervention du développeur dans le choix et la composition des Services Web. Ce type de composition est la composition dynamique. **Microsoft Biztalk** et **Bea Weblogic** sont deux exemples de moteurs de composition statique de Services Web [18]. Si les fournisseurs de services proposent d'autres services ou changent les anciens services, des incohérences peuvent être causées, ce qui demanderait un changement de l'architecture du logiciel, voire de la définition du processus et créerait l'obligation de faire une nouvelle conception du système. Dans ce cas, la composition statique des Services Web est considérée trop restrictive : les composants doivent s'adapter automatiquement aux changements imprévisibles.

2.3.2 Composition dynamique des Services Web

Une composition est dynamique quand les Services Web qui font partie de la composition sont connus progressivement, ainsi que l'ordre d'exécution nécessaire pour la composition. Une approche dynamique pour la composition de services offre le potentiel de réaliser des applications flexibles et adaptables, en sélectionnant et en combinant les services de manière appropriée sur la base de la requête et du contexte de l'utilisateur [7].

Dans cette approche, les Services Web à composer sont choisis au moment de l'exécution durant laquelle une recherche des services est effectuée dans les registres. À titre d'exemple, **StarWSCOP** [8] [18]. Pour faire la composition dynamique des Services Web, StarWSCOP effectue les quatre étapes suivantes :

- Les fournisseurs des Services Web publient leurs services dans un registre.
- StarWSCOP décompose les requêtes des utilisateurs en services abstraits et envoie une requête SOAP au registre pour trouver les services appropriés.
- Le registre de Services Web disponibles renvoie une liste de Services Web concrets.
- StarWSCOP envoie une requête SOAP aux Services Web trouvés, ensuite se lie avec eux.

StarWSCOP contient plusieurs modules : un système intelligent pour décomposer les requêtes des utilisateurs en plusieurs services abstraits, un moteur de recherche de Services Web pour découvrir les Services Web qui respectent les conditions de l'utilisateur, un moteur de composition qui exécute les services en ordre. Une couche axée ontologie est aussi rajoutée à UDDI afin de faire de l'appariement sémantique pour les Services Web.

2.4 Réécriture de requêtes en termes de vues

Le problème de réécriture de requêtes a été principalement abordé dans les applications d'optimisation de requêtes et d'intégration de données. La principale question à laquelle la réécriture essaie de répondre est : Etant donné une requête Q est un ensemble de vues $V = \{V_1, ..., V_n\}$, est-il possible de répondre à Q en utilisant uniquement les données des

8. Star Web Services Composition Platform

vues V ?. Dans le contexte d'intégration de données, la réécriture d'une requête consiste à déterminer les vues pertinentes pour l'exécution de la requête d'utilisateur et à utiliser leurs définitions pour reformuler cette requête. Les techniques de réécriture de requêtes ont été largement explorées dans le domaine de base de données. Une bonne étude des algorithmes de réécriture de requête est présentée dans [2].

Dans ce qui suit, nous décrivons trois algorithmes de réécriture permettant de répondre aux requêtes en utilisant des vues. Ces algorithmes sont : ***"l'algorithme Bucket"***, ***"l'algorithme de règles inversées"*** et ***"l'algorithme MiniCon"***.

2.4.1 L'algorithme Bucket [2]

L'idée principale de l'algorithme Bucket est de réduire le nombre de réécritures qui doivent être étudiées en considérant chaque sous-but de la requête en isolation afin de déterminer les vues pertinentes pour chaque sous-but. L'algorithme procède en deux étapes : (i) construction d'un bucket pour chaque sous-but de la requête contenant les vues contributives pour la réécriture de ce sous-but , et (ii) construction de toutes les requêtes conjonctives possibles en prenant une source de chaque bucket. Le résultat final de l'algorithme est l'union de toutes les requêtes conjonctives (réécritures candidates).

Dans sa première phase, l'algorithme Bucket essaie de trouver l'ensemble des vues qui sont pertinentes pour chaque sous-but de la requête. Une vue V est pertinente pour un sous-but g d'une requête Q si les conditions suivantes sont vérifiées :

- V contient un sous-but g_1 tel que g peut être unifié avec g_1. L'unification est faite sur des sous-buts ayant le même nom en tenant compte de la position des variables. Elle consiste à construire les mappings entre les variables des deux sous-buts ;
- Toutes les variables distinguées de Q sont mappées à des variables distinguées de V lors de l'unification de g et g_1 ;
- Les prédicats de comparaison de V et de Q ne sont pas conflictuels.

Dans la deuxième phase, l'algorithme Bucket trouve un ensemble de réécritures de requêtes conjonctives. Il considère toutes les combinaisons possibles de sources de don-

nées en prenant une source de chaque bucket. Chacune de ces réécritures représente un moyen d'obtenir une partie de la réponse à Q à partir des vues. L'inconvénient majeur de l'algorithme Bucket est son incapacité de détecter le fait qu'il doit utiliser la même vue pour couvrir plusieurs sous-buts de la requête. Cet algorithme est également incapable de trouver les vues qui ne peuvent pas être pertinentes avant d'avoir testé toutes les solutions possibles.

2.4.2 L'algorithme de règles inversées [2]

Le fonctionnement de l'algorithme de règles inversées est basé sur la construction d'un ensemble de règles qui montrent comment construire les tuples du schéma global à partir des tuples des vues. Le principe de construction des règles inversées est le suivant :

- Pour chaque vue V_i définie par l'expression $V_i(\overline{X}) : - r_1(\overline{X}_1)$ ...., $r_n(\overline{X}_n)$., construire n règles telles que l'entête de la règle R_j j=1... n, est l'atome $r_j(\overline{X}_j)$ du corps de V_i et le corps de R_j est l'entête de la vue $V_i(V_i(\overline{X}))$.
- Remplacer toutes les variables existentielles de la définition de V_i qui se retrouvent dans les entêtes des règles inversées par des fonctions de Skolem. Utilisez la même fonction de Skolem pour toutes les occurrences de la même variable existentielle de V_i. Les fonctions de Skolem indiquent les variables dont les valeurs sont inconnues en dehors de la définition de la vue.
- L'ensemble des règles inversées est l'union des règles produites pour chaque vue.

L'avantage de l'approche de réécriture de requêtes à base de règles inversées est la construction des règles qui peut être fait en temps polynomial. Par contre, le calcul des résultats de la requête peut répéter des calculs qui ont été faits pour le calcul des vues. Un autre inconvénient de cette approche est que lors de calcul des tuples, l'algorithme va utiliser l'ensemble des règles inversées, même si certaines règles ne sont pas pertinentes à la réécriture de la requête i.e., il n'est pas possible d'obtenir des résultats pour la requête en utilisant ces règles.

2.4.3 L'algorithme MiniCon [2]

Comme l'algorithme Bucket, l'algorithme MiniCon fonctionne en deux étapes qui sont la recherche des vues pertinentes pour la réécriture de la requête et la combinaison de ces vues afin d'obtenir les réécritures de la requête. Dans la première phase, il cherche des correspondances entre les sous-buts de la requête et ceux des vues. A la différence de l'algorithme Bucket qui construit des tas indépendants pour chaque sous-but, lorsque l'algorithme MiniCon détecte une correspondance entre un sous-but g de la requête Q et un sous-but g_1 d'une vue V, il examine les variables de jointure de la requête afin de déterminer l'ensemble minimal de sous-buts de V qui doivent être unifiés avec des sous-buts de Q étant donné que g sera unifié avec g_1. Pour chaque vue, l'algorithme note les sous-buts de la requête qui peuvent être couverts par cette vue. L'ensemble des informations décrivant les mappings entre une vue V et une requête Q sont représentés par un quadruplet $(h, V(\overline{Y}), \Phi, G)$ appelé MiniCon Descriptor (MCD) où :

- h est un homomorphisme défini sur les variables de l'entête de V. Il exprime le fait que parfois il peut être nécessaire d'unifier des variables de l'entête de la vue.
- $V(\overline{Y})$ est le résultat de l'application de h sur l'entête de V.
- Φ est le mapping (unification) partiel des variables de Q vers V.
- G est l'ensemble de sous-buts de Q qui sont couverts par la vue V.

Une fois les MCDs construits, l'algorithme les combine afin d'obtenir des réécritures de la requête. La combinaison de MCDs est faite en suivant le principe que l'ensemble des MCDs formant une réécriture candidate de la requête doivent couvrir l'ensemble des sous-buts de la requête et que chaque paire de MCDs doivent couvrir des sous-buts disjoints i.e. : soient $C_1, C_2, \ldots, C_k$ un ensemble de MCDs formant une réécriture candidate de Q. Alors, les conditions suivantes doivent être vérifiées :

- $G_1 \cup \cdots \cup G_k = \text{sous-buts } (Q)$ et
- $\forall i, j, i \neq j, G_i \cap G_j = \emptyset$

Le principal avantage de l'algorithme MiniCon est sa capacité de filtrer les vues non pertinentes pendant la phase de construction des MCDs, les MCDs construits peuvent être combinés sans vérifier la satisfaisabilité des prédicats de jointure.

2.5 Discussion

Nous avons présenté les travaux concernant deux manières de réifier l'intégration de données :

- En se basant sur des techniques de réécriture des requêtes en termes de vues ;
- En se basant sur des plates-formes permettant une composition des données accessibles à distance de manière statique ou dynamique.

Pour la première, ces techniques de réécriture ont été appliquées sur les premiers systèmes de médiation réalisés le plus souvent pour interroger de manière uniforme des bases de données distribuées.

Pour la seconde manière d'intégrer l'information, beaucoup des plates-formes utilisées sont industrielles et se basent sur des Services Web DaaS. Nous donnons à titre d'exemple les produits mis au point pour faire la création de Services Web DaaS : **AquaLogic Data Services Platform (ALDSP)** par BEA Systems [8], **Astoria** par Microsoft [10], la plateforme **MetaMatrix** par RedHat [16], **Composite Data Virtualization Platform** par Composite Software [9] et la plateforme **XIC (Xcalia Intermediation Core)** par Xcalia [24].

Le système que nous allons proposé et présenté dans la suite, s'inscrit dans le cadre de travaux issus de ces deux manières d'intégrer l'information. Notamment, notre cadre permet de combiner des techniques de réécriture de requêtes en termes de vues représentées sous forme de Services Web DaaS. Ces deux classes d'intégration ont été considérées le plus souvent de façon isolée et très peu de travaux les ont appliquées dans un même cadre de médiation. Nous pouvons cités les travaux dans les deux thématiques, les travaux de Mahmoud Barhamgi [6]. Cependant, son algorithme de réécriture ne fait aucune référence aux algorithmes classiques de réécriture de requêtes contrairement à notre travail. En outre, notre approche traite sans ambiguïté tous les cas possibles contrairement à

l'approche de Barhamgi [6] qui laisse dans un certains nombre de cas certaines questions sans réponses.

Dans la suite, nous allons présenter notre approche de composition de Services Web DaaS en se basant sur des techniques de réécriture de requêtes.

Chapitre 3

Conception de notre système de médiation

Sommaire

3.1 Introduction

Dans ce chapitre, nous nous intéressons à la conception de notre système de médiation. Tout d'abord, nous commençons par présenter l'architecture globale du système dans la section 3.2. Ensuite, nous décrivons dans la section 3.3 un scénario de motivation

pour l'interrogation et la composition des Services Web DaaS, nous identifions aussi les principaux enjeux liés à la composition et nous décrivons notre approche de réécriture de requête pour la composition des Services Web DaaS. Dans la section 3.4, nous présentons notre ontologie de médiation, notre modèle de requête proposé ainsi que les vues RDF paramétrées parmettant de décrire nos services DaaS proposés. La section 3.5 est consacrée à l'enrichissement des vues RDF en ajoutant les contraintes sémantiques RDFS. Enfin, nous présentons notre algorithme de réécriture de requête permettant la sélection, l'invocation et la composition des Services Web DaaS afin de répondre aux requêtes d'utilisateur (section 3.6).

3.2 Architecture globale du système

L'architecture du système a été divisée en deux parties : une partie pour l'interrogation et l'autre pour l'indexation (Figure 3.1). Notre travail s'appui sur la partie d'interrogation dans laquelle l'utilisateur voit le système comme une seule interface du Service Web basée sur un système intermédiaire qui est le **système médiateur**. Ce dernier joue le rôle d'interface entre l'utilisateur et les sources d'information en donnant l'impression à l'utilisateur qu'il interroge un seul système *centralisé* et *homogène* alors que les sources interrogées sont réparties, autonomes et hétérogènes.

Le système médiateur est fondé sur la définition d'un *schéma global* qui fournit un vocabulaire unique pour l'expression des requêtes des utilisateurs et pour la description des contenus des sources par un ensemble de vues. Il offre à l'utilisateur une vue uniforme des sources de données qu'il exploite et permettre de les interroger d'une manière transparente sans que l'utilisateur n'ait souci de la provenance des informations ni de leur format d'origine. L'interrogation des sources de données est effectuée par le service d'interrogation QAS (Quering As a Service).

L'utilisateur exprime sa requête en utilisant le vocabulaire fourni par une ontologie. La requête posée sur le schéma global nécessite une combinaison des Services Web DaaS et de ce fait, elle doit être reformulée par le médiateur en des sous requêtes sur les sources de données (ce qui est nécessaire en raison du fait que le schéma global lui-même ne contient aucune donnée). Le problème de la reformulation de la requête (Rewriting) est

connu comme étant le problème de réécriture de requête en termes de vues pour lequel nous proposons un algorithme qu'on va le détailler par la suite.

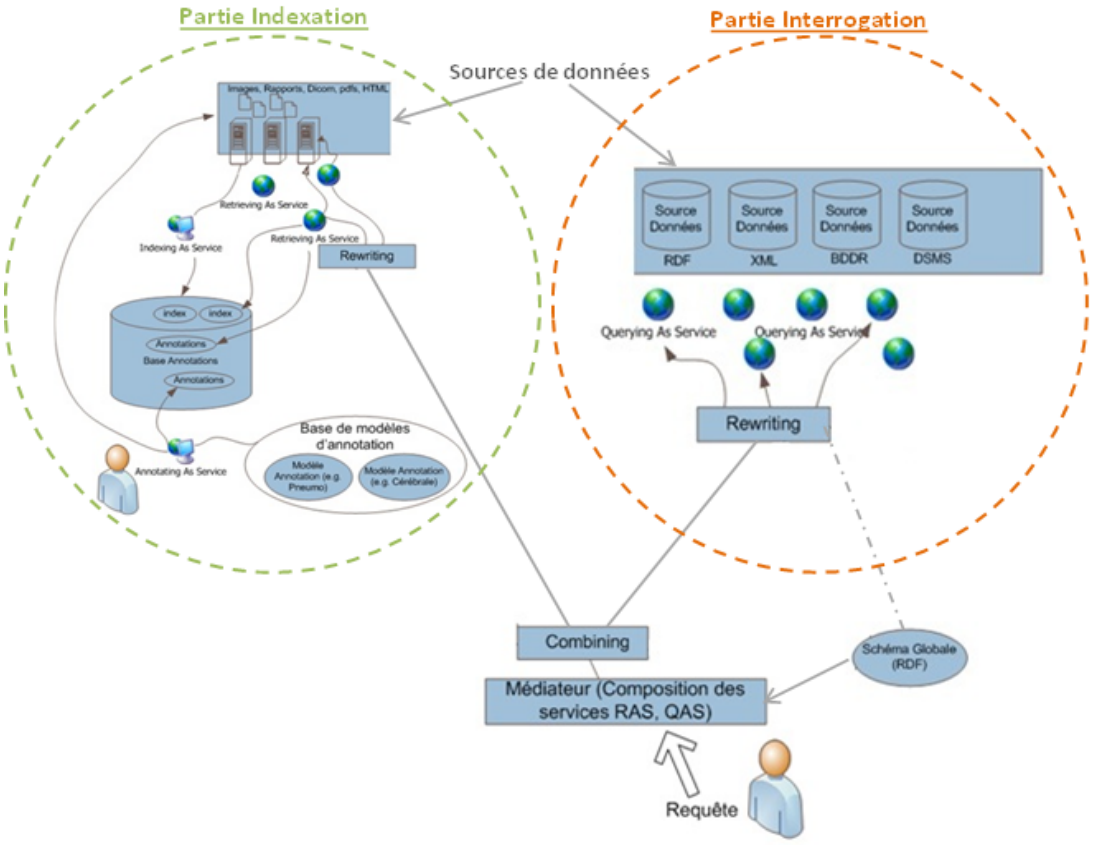

FIGURE 3.1 – Architecture globale du système

3.3 Exemple de motivation

Un Service Web DaaS peut fournir des informations intéressantes ; mais dans la plupart des cas, les requêtes des utilisateurs exigent l'invocation de plusieurs services. Supposons que le médecin Yasmin a la requête suivante Q : " Donnez Les rapports fournis par le médecin 'p100' des patients souffrant de la maladie du diabète identifiée par le code 'D5' et qui sont suivis par des infirmiers travaillant dans le service 'S12' " .

En effet, Yasmin dispose d'un ensemble de Services Web DaaS représentés dans la table 3.1

Services	**Fonctionnalité**	**Contraintes**
$S_1(\$a, ?b)$	Donne les rapports (b) fournis par un médecin (a)	$a >= p150$
$S_2(\$a, ?b)$	Donne les rapports (b) fournis par un médecin (a)	$a < p150$
$S_3(\$a, ?b)$	Donne les patients (b) souffrant d'une maladie (a)	
$S_4(\$a, ?b)$	Donne les infirmiers (b) travaillant dans un service (a)	a>S18
$S_5(\$a, ?b)$	Donne les infirmiers (b) travaillant dans un service (a)	a<=S18
$S_6(\$a, ?b)$	Donne les patients (b) suivis par un infirmier (a)	
$S_7(\$a, ?b)$	Donne les rapports (b) d'un patient (a)	
$S_8(\$a, ?b)$	Donne les médecins (b) traitant un patient (a)	
$S_9(\$a, ?b)$	Donne les médecins (b) travaillant dans un service (a)	
$S_{10}(\$a, ?b)$	Donne les rapports (b) liés à un rapport (a)	
$S_{11}(\$a, ?b, ?c, ?d)$	Donne les pièces jointes images (b), vidéos (c), dicom (d) d'un rapport (a)	

TABLE 3.1 – Services Web DaaS proposés

Evidemment, Yasmin utilise ces services pour obtenir une réponse à sa requête. Comme la montre la figure 3.2, Yasmin invoque le service S2 pour trouver la liste (L1) des rapports fournis par le médecin Youcef (étape1), ça veut dire que le code de notre médecin satisfait la contrainte 'a<p150'. Dans la deuxième étape, elle invoque le service S5 puisque le code du service concerné satisfait la contrainte 'a<=S18' et cela pour trouver la liste (L2) des infirmiers travaillant dans le service 'S12'. Notons que les étapes 1 et 2 peuvent être exécutées en parallèle. Ensuite, elle invoque le service S6 pour chaque infirmier de la liste (L2) pour obtenir la liste des patients (L3) qui les suit. Après cette étape, elle invoque le service S3 pour chaque patient de la liste L3 afin de trouver la liste L4 contenant les maladies dont il souffre. Dans l'étape 5, elle va filtrer la liste L4 en prenant les tuples dont la deuxième partie fait référence à 'D5' ce qui nous donne la liste (L5). Yasmin va invoquer le service S7 pour chaque patient de L5 pour trouver les rapports qui lui concerne (étape 6), le résultat de cette étape est mis dans la liste L6. Enfin, Yasmin va effectuer une intersection entre les listes L1 et L6 et donc, elle obtient une réponse à sa requête posée.

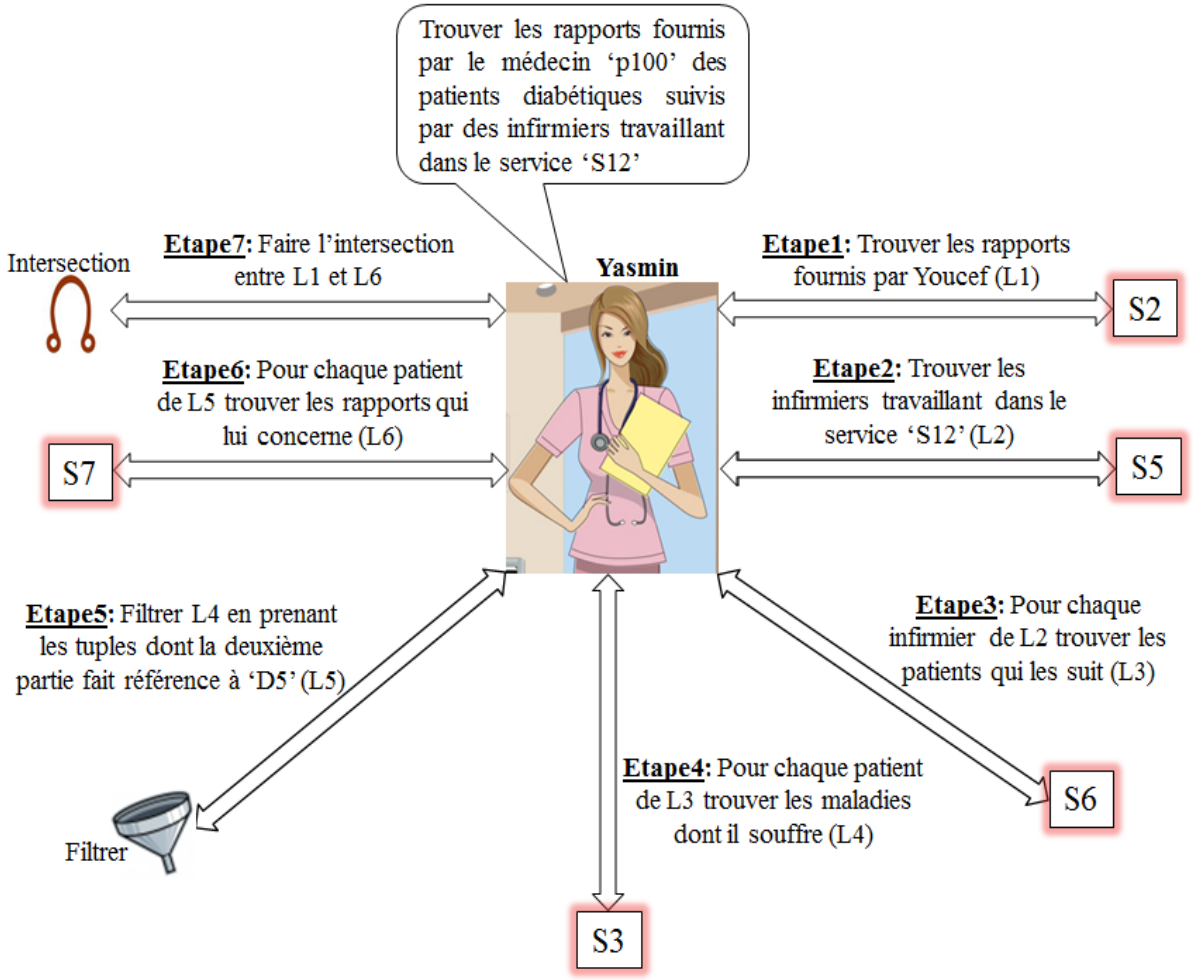

FIGURE 3.2 – Scénario de motivation proposé

- **Les enjeux**

Notre scénario montre que Yasmin a besoin d'effectuer plusieurs tâches pour exécuter sa requête. Ces tâches peuvent notamment être fastidieuses lorsque le nombre de services est important. Premièrement, Yasmin a besoin de comprendre la sémantique des services DaaS existants et les relations entre les paramètres d'entrée et de sortie pour chaque service. Deuxièmement, elle a besoin de sélectionner manuellement les services qui sont pertinents à sa requête et les invoquer dans le bon ordre. Elle doit aussi comprendre le plan d'exécution pour sa requête. Troisièmment, Yasmin a besoin de consolider les résultats et effectuer manuellement des jointures potentielles ou un filtrage des résultats comme mentionné dans les étapes 5 et 7 du scénario.

- **Aperçu de l'approche proposée**

Nous proposons une approche de réécriture de requêtes pour la composition automatique de Services Web DaaS comme la montre la figure 3.3. Elle suppose l'existence d'une ontologie de médiation pour capturer les connaissances consensuelles et partagées dans un domaine donné (domaine médical dans notre cas). Les Services Web DaaS sont modélisés par des vues RDF à partir de l'ontologie de médiation. Les vues RDF capturent les relations sémantiques entre les paramètres d'entrée et de sortie en utilisant les concepts ontologiques définis dans l'ontologie de médiation, elles sont enregistrées dans l'annuaire des services.

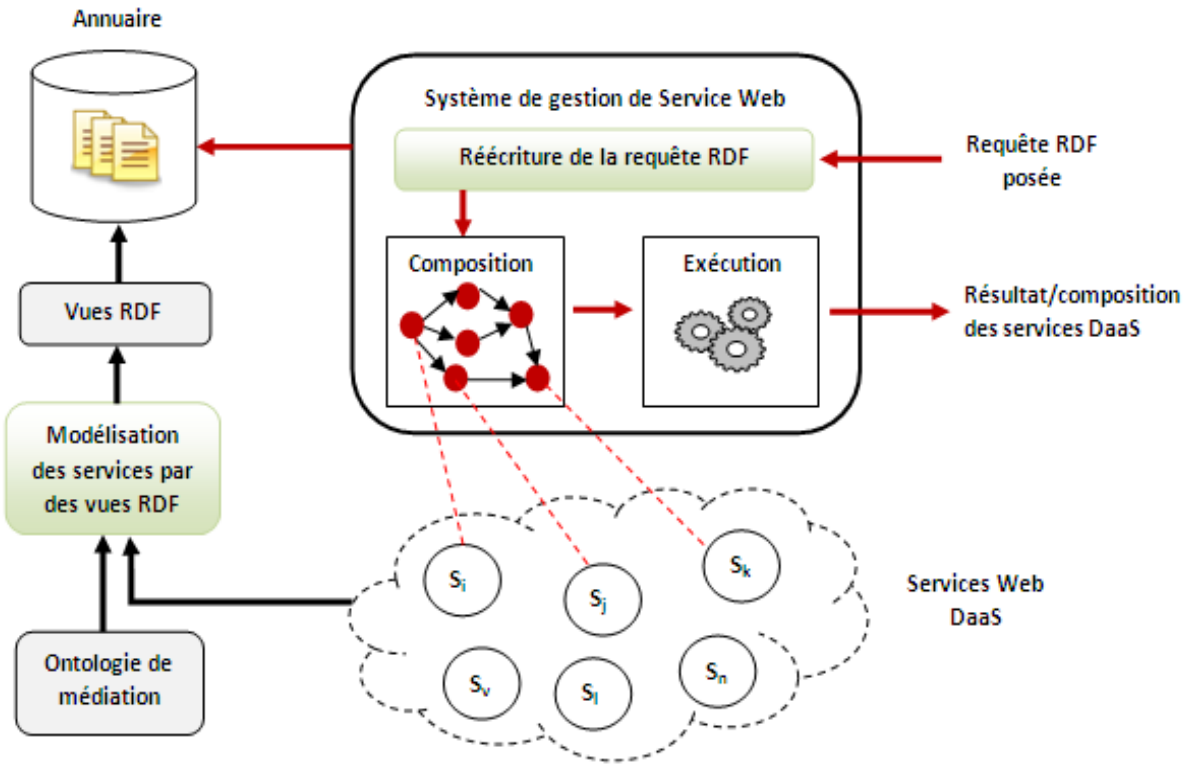

FIGURE 3.3 – Aperçu de l'approche proposée

Les utilisateurs posent leurs requêtes en utilisant le langage de requête SPARQL. Le WSMS (Web Service Management System) utilise un module de réécriture de requête RDF et les vues RDF existantes pour sélectionner les services qui peuvent être combinés pour répondre à la requête posée. Ensuite, il génère un service composite, il l'exécute et il retourne les données demandées à l'utilisateur. Le service composite généré peut être aussi déployé comme un nouveau service DaaS et utilisé pour répondre à d'autres requêtes.

3.4 Services DaaS et modèle de requête

3.4.1 Ontologie de médiation

Dans l'approche proposée, les utilisateurs formulent leurs requêtes à partir d'une ontologie de médiation. Cette dernière est décrite en RDF/RDFS. Formellement, une ontologie de médiation **OM** est définie par 6 tuples (C,D, TP, OP, SC, SP) où :

- **C** est l'ensemble de Classes.
- **D** est l'ensemble de DataTypes.
- **DP** est l'ensemble de DataType Properties. Chaque DataType Property a un domaine dans **C** et un co-domaine (range) dans **D**.
- **OP** est l'ensemble des Object Properties. Chaque Object Property a son domaine et co-domaine dans **C**.
- **SC** est une relation dans (**C** × **C**), représentant les relations `sub-class of` entre les classes. Par exemple, C_2 **SC** C_1 signifie que C_2 est une sous classe de C_1.
- **SP** est une relation dans [(**OP** × **OP**) ∪ (**DP** × **DP**)]. Elle représente les relations `sub-property of` entre les Properties dans OP ou dans DP. Par exemple, p_2 **SP** p_1 signifie que p_2 est une sous propriété de p_1.

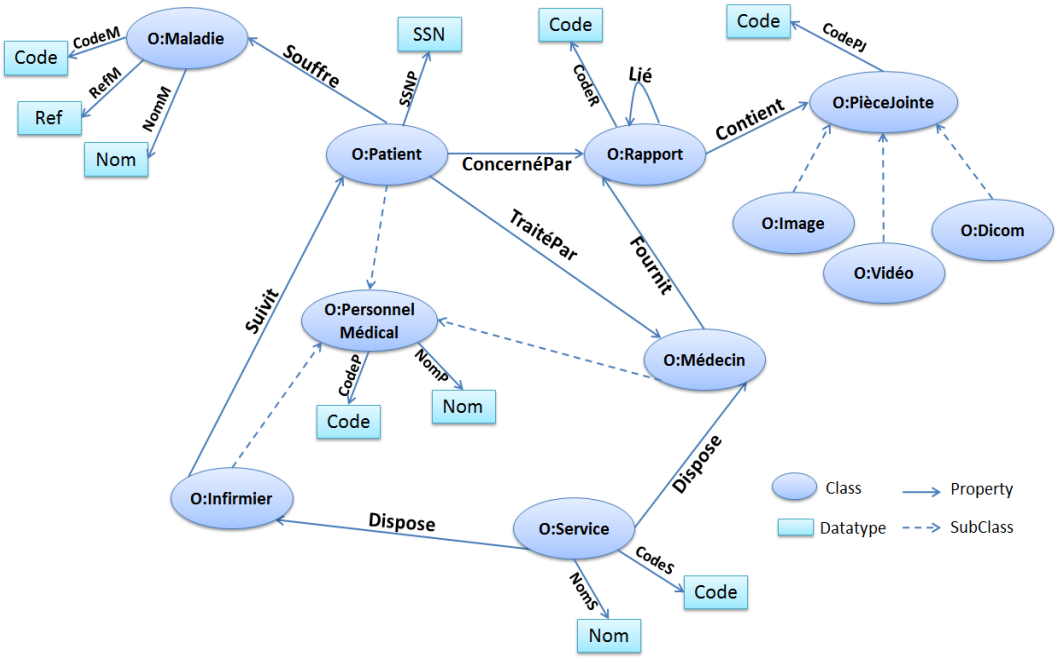

FIGURE 3.4 – Ontologie de médiation proposée

La figure 3.4 représente notre ontologie de médiation proposée concernant le domaine médical. Prenons les trois concepts Patient, Médecin et Service, notre ontologie spécifie que le patient est traité par un médecin et ce dernier travaille dans un service. Un Service a deux caractéristiques (DataTypes) qui sont le *Code* et le *Nom*. Les concepts sont reliés entre eux par des Object Properties et aux DataTypes via les DataType Properties.

3.4.2 Requête

On considère les requêtes conjonctives à partir d'une ontologie de médiation. Les requêtes sont exprimées en utilisant SPARQL. Formellement une requête Q a la forme suivante :

$$Q(\overline{X}) :- G(\overline{X}, \overline{Y}) \ où$$

- $Q(\overline{X})$ est appelé l'entête de la requête, elle a la forme d'un prédicat relationnel.
- $\overline{X}$ sont appelées les variables de l'entête ou `les variables distinguées`.
- $\overline{Y}$ sont appelées `les variables existentielles`.
- $G(\overline{X}, \overline{Y})$ est appelé le corps de la requête, et est un ensemble de triplets RDF où chacun a la forme `sujet.propriété.objet`. Le corps peut également contenir des contraintes sur les variables du corps de la forme : x Θ CONSTANT, où : $\Theta \in \{>,\geq,<,\leq\}$.

Les deux figures 3.5 et 3.6 montrent la représentation graphique et SPARQL de la requête Q : " Donnez Les rapports fournis par le médecin 'p100' des patients souffrant de la maladie du diabète identifiée par le code 'D5' et qui sont suivis par des infirmiers travaillant dans le service 'S12' " .

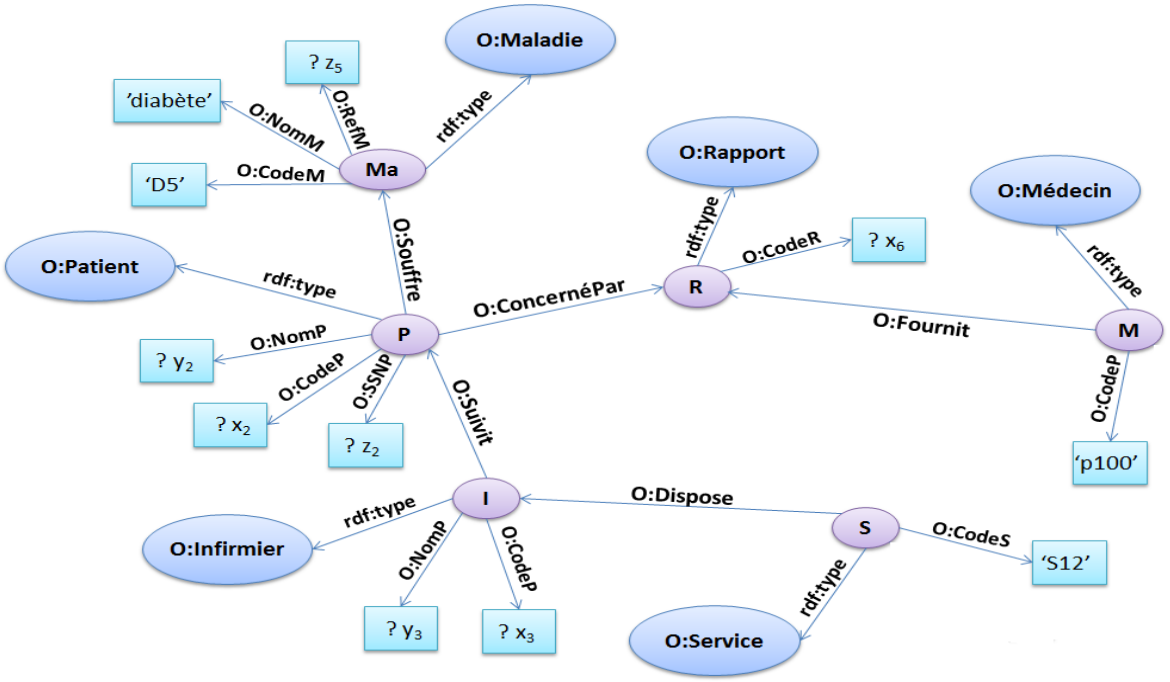

FIGURE 3.5 – Représentation graphique de la requête Q

$Q(y_1,x_2,y_2,z_2,x_3,y_3,y_4,z_5,x_6)$:-

?M	.rdf:type	.O:Médecin
?M	.O:CodeP	.'p100'
?M	.O:Fournit	.?R
?P	.rdf:type	.O:Patient
?P	.O:CodeP	.?x_2
?P	.O:NomP	.?y_2
?P	.O:SSNP	.?z_2
?P	.O:ConcernéPar	.?R
?P	.O:Souffre	.?Ma
?I	.rdf:type	.O:Infirmier
?I	.O:CodeP	.?x_3
?I	.O:NomP	.?y_3
?I	.O:Suivit	.?P
?S	.rdf:type	.O:Service
?S	.O:CodeS	.'S12'
?S	.O:Dispose	.?I
?Ma	.rdf:type	.O:Maladie
?Ma	.O:CodeM	.'D5'
?Ma	.O:NomM	.'diabète'
?Ma	.O:RefM	.?z_5
?R	.rdf:type	.O:Rapport
?R	.O:CodeR	.?x_6

FIGURE 3.6 – Représentation SPARQL de la requête Q

Une requête peut être vue comme un graphe avec deux types de noeuds :

- **Noeuds Classe** : un noeud classe se réfère aux classes dans l'ontologie de médiation (exemple : P et R). Ils sont reliés via les Object Properties et représentent les variables exitentielles de la requête.

- **Noeuds Littéraux** : ils représentent les DataTypes (exemple : x_2, y_2,z_2). Ils sont liés avec les noeuds classe via les DataType Properties. Les noeuds littéraux peuvent correspondre aux deux variables existentielles et distinguées.

3.4.3 Vue RDF Paramétrée (RPV)

Nous modélisons les Services DaaS par des RPVs [9] à partir de l'ontologie de médiation. Les RPVs utilisent les concepts et les relations de l'ontologie de médiation pour capturer la relation sémantique entre l'entrée et la sortie. Une RPV exige un ensemble particulier des entrées afin de récupérer un ensemble particulier des sorties. Ces dernières ne peuvent pas être récupérées sauf si les entrées sont liées.

Une RPV peut être vue comme une requête SPARQL paramétrée. Formellement, une RPV d'un Service DaaS S_i à partir d'une ontologie de médiation est un prédicat

$$S_i(\overline{X}_i, \overline{Y}_i) :- \ < \Phi(\overline{X}_i, \overline{Y}_i, \overline{Z}_i), Ct_i >, \ où :$$

- $\overline{X}_i$ est l'ensemble des variables indiquant les paramètres d'entrée nécessaires pour invoquer S_i , elles sont appelées *les variables d'entrée.*
- $\overline{Y}_i$ est l'ensemble des variables désignant les littéraux retournés après l'invocation du S_i, elles sont appelées *les variables de sortie*. Les variables d'entrée et de sortie sont également appelées les `variables distinguées`.
- $\Phi(\overline{X}_i, \overline{Y}_i, \overline{Z}_i)$ représente la relation sémantique entre les variables d'entrée et de sortie. $\overline{Z}_i$ est l'ensemble des `variables existentielles` reliant $\overline{X}_i$ et $\overline{Y}_i$. Φ a la forme de triplets RDF où chaque triplet a la forme `sujet. propriété.objet`.
- Ct_i sont les contraintes imposées sur les variables $\overline{X}_i$, $\overline{Y}_i$ ou $\overline{Z}_i$. Une contrainte a la forme : Θ CONSTANT, où : $x\ \Theta \in \{>,\geq,<,\leq\}$

9. RDF Parameterized Views

Nous présentons ci-dessous les RPVs de nos services DaaS représentés dans le Tableau 3.1. Chaque RPV est caractérisée par un modèle d'accès ; ce dernier spécifie les paramètres d'entrée et de sortie. Les variables d'entrée et de sortie sont préfixées respectivement par les symbôles "**$**" et "**?**".

Les services $S1$ et $S2$ donnent les rapports (b) fournit par un médecin (a).

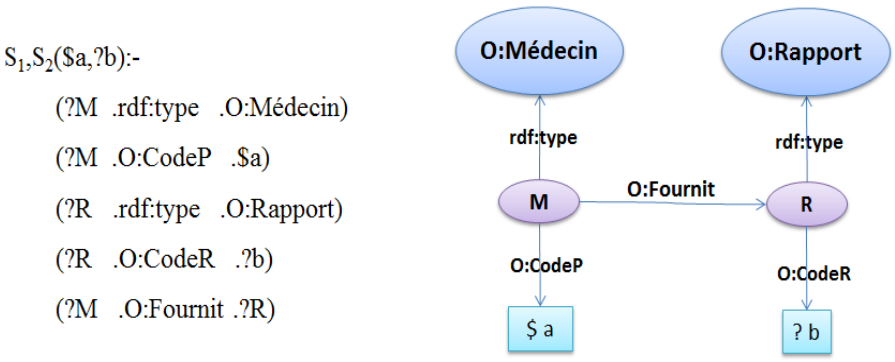

FIGURE 3.7 – Graphe RDF des services $S1$ et $S2$

Le service $S3$ donne les patients (b) souffrant d'une maladie (a).

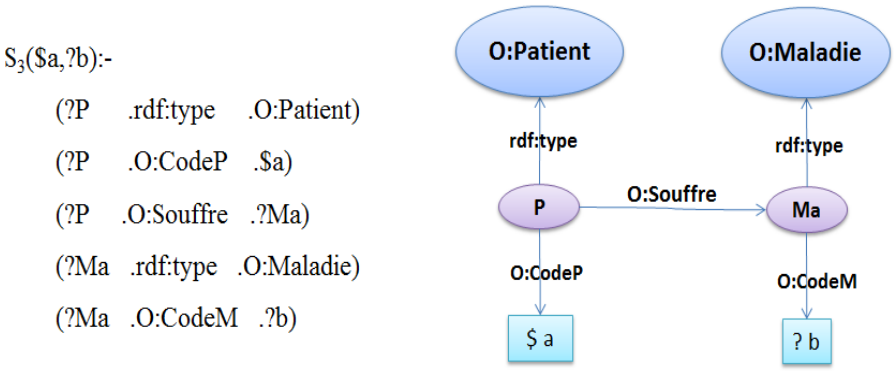

FIGURE 3.8 – Graphe RDF du service $S3$

Les services $S4$ et $S5$ donnent les infirmiers (b) travaillant dans un service (a).

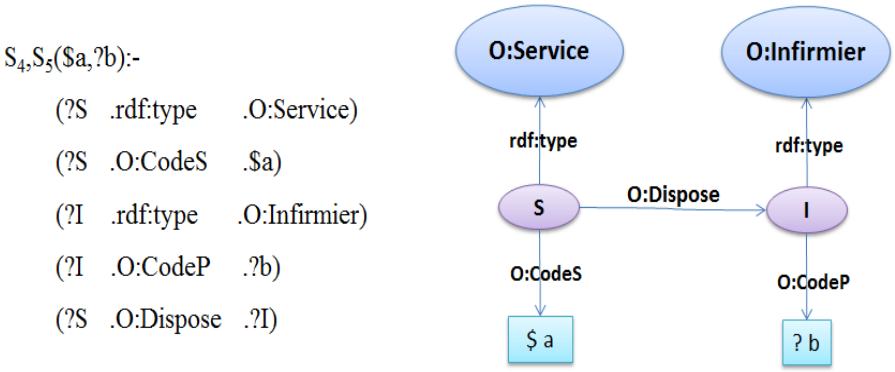

FIGURE 3.9 – Graphe RDF des services $S4$ et $S5$

Le service $S6$ donne les patients (b) suivi par un infirmier (a).

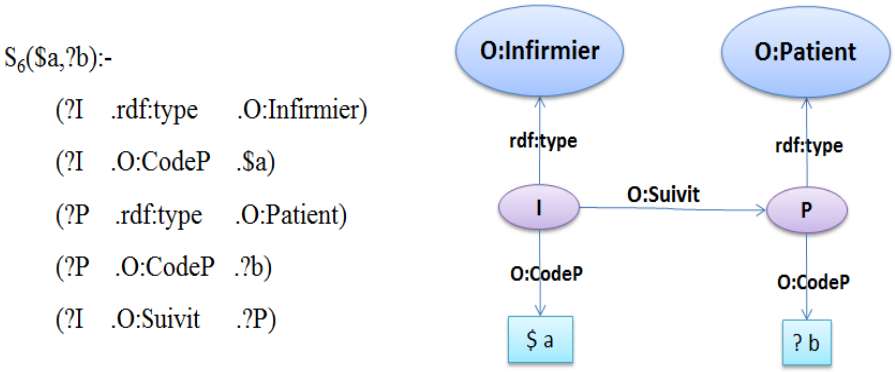

FIGURE 3.10 – Graphe RDF du service $S6$

Le service $S7$ donne les rapport (b) d'un patient (a).

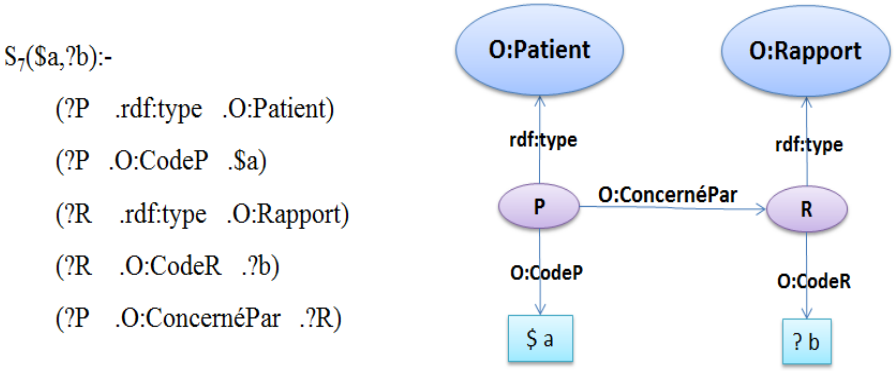

FIGURE 3.11 – Graphe RDF du service $S7$

Le service $S8$ donne les médecins (b) traitant un patient (a).

$S_8(\$a,?b)$:-

(?P .rdf:type .O:Patient)
(?P .O:CodeP .$a)
(?M .rdf:type .O:Médecin)
(?M .O:CodeP .?b)
(?P .O:TraitéPar .?M)

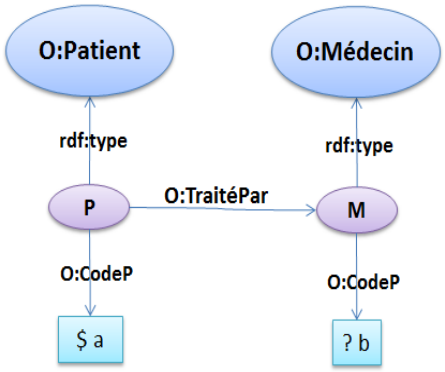

FIGURE 3.12 – Graphe RDF du service $S8$

Le service $S9$ donne les médecins (b) travaillant dans un service (a).

$S_9(\$a,?b)$:-

(?S .rdf:type .O:Service)
(?S .O:CodeS .$a)
(?M .rdf:type .O:Médecin)
(?M .O:CodeP .?b)
(?S .O:Dispose .?M)

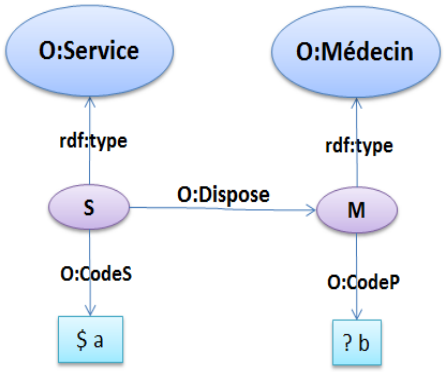

FIGURE 3.13 – Graphe RDF du service $S9$

Le service $S10$ donne les rapports (b) liés à un rapport (a).

$S_{10}(\$a,?b)$:-

($?R_1$ .rdf:type .O:Rapport)
($?R_1$ .O:CodeR .$a)
($?R_2$ .rdf:type .O:Rapport)
($?R_2$ .O:CodeR .?b)
($?R_1$ .O:Lié .$?R_2$)

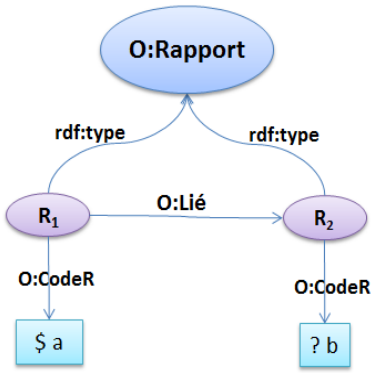

FIGURE 3.14 – Graphe RDF du service $S10$

Le service $S11$ donne les pièces jointes images (b), vidéos (c), dicom (d) d'un rapport (a).

S_{11}($a,?b,?c,?d):-
(?R .rdf:type .O:Rapport)
(?R .O:CodeR .$a)
(?R .O:Contient .?Pj_1)
(?R .O:Contient .?Pj_2)
(?R .O:Contient .?Pj_3)
(?Pj_1 .rdf:type .O:Image)
(?Pj_1 .O:CodePJ .?b)
(?Pj_2 .rdf:type .O:Vidéo)
(?Pj_2 .O:CodePJ .?c)
(?Pj_3 .rdf:type .O:Dicom)
(?Pj_3 .O:CodePJ .?d)

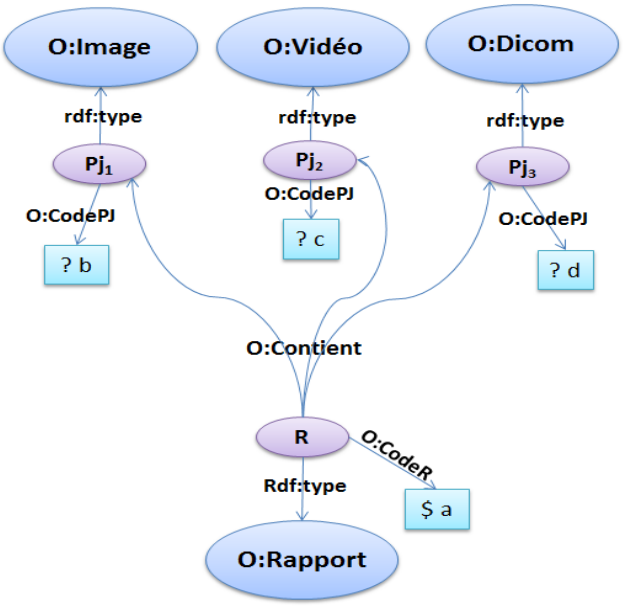

FIGURE 3.15 – Graphe RDF du service $S11$

3.5 Pré-traitement des vues RDF

Les vues RDF associées aux services DaaS sont pré-traitées avant la résolution des requêtes. Le pré-traitement permet au médiateur de renvoyer plus de résultats pour une requête donnée. La phase de pré-traitement permet de surmonter le problème basé sur les contraintes sémantiques de sous-classement qui sont définis dans l'ontologie. Nous identifions deux étapes de pré-traitement qui sont : (i) l'application des contraintes sémantiques RDFS et (ii) l'association des fonctions Skolem aux Noeuds Classe.

- **Application des contraintes sémantiques RDFS**

Dans cette étape, les vues RDF sont étendues à prendre en compte les contraintes sémantiques RDFS de l'ontologie médiation. Les contraintes sémantiques RDFS inclus : *"rdfs:subClassOf", "rdfs:subPropertyOf", "rdfs:domain" et "rdfs:range".*

Prenons l'exemple de la figure 3.16 (Partie-A), la contrainte RDFS "O:Homme rdfs:sub ClassOf O:Personne" est utilisée pour étendre la vue RDF avec un nouveau triplet RDF pour indiquer qu'une variable de type"Homme" est aussi de type "Personne" ; les contraintes

RDFS du domaine et du range qui sont définies sur la propriété "O:Parent" (i.e. " O:Parent rdfs:domain O:Homme " et " O:Parent rdfs:range O:Fils ") sont aussi utilisées pour étendre la vue dans la figure 3.16 (Partie-B) avec les types d'information pour les variables H et F.

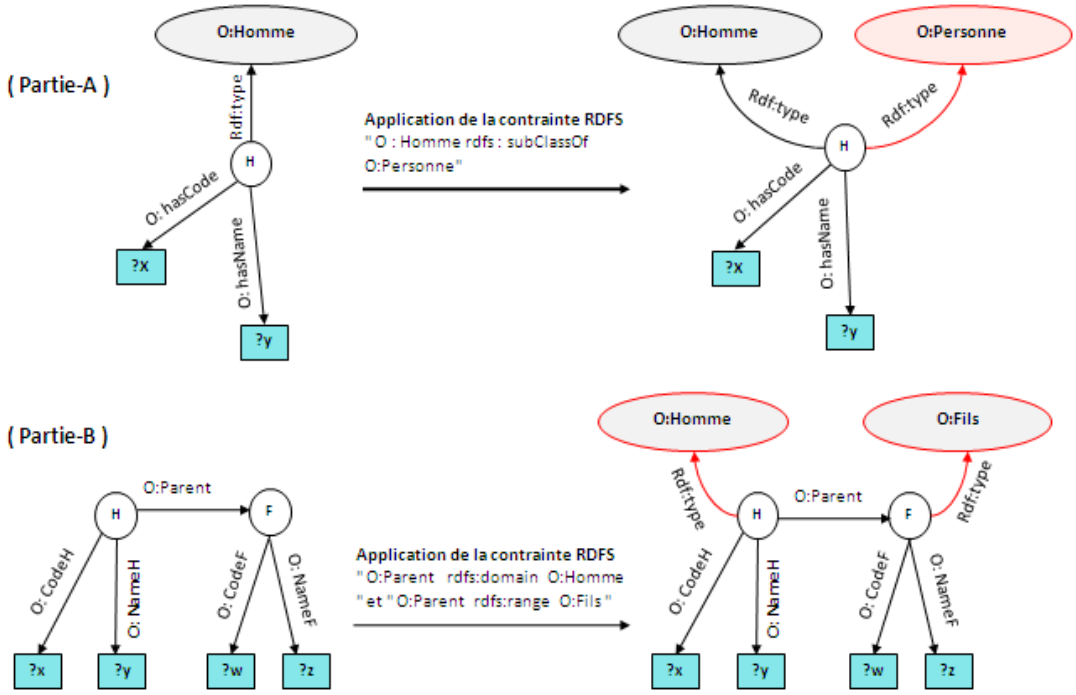

FIGURE 3.16 – Application des contraintes sémantiques RDFS

Voici ci-dessous l'application des contraintes sémantiques RDFS sur les RPVs.

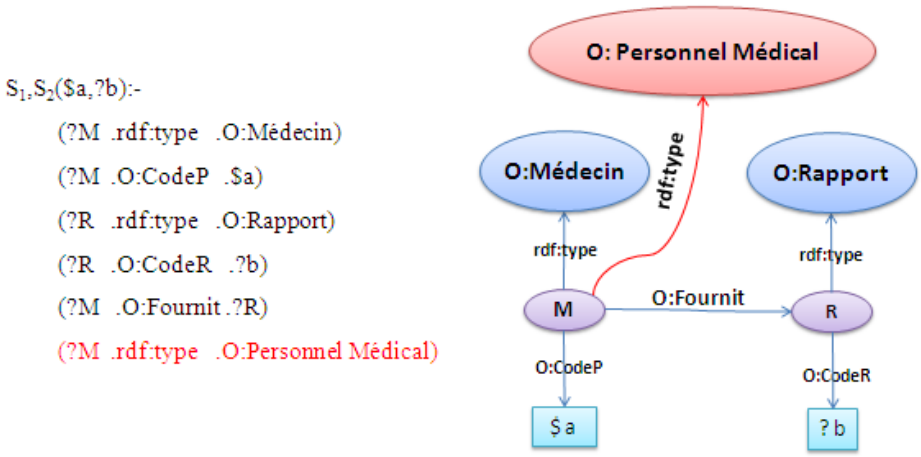

FIGURE 3.17 – Graphe RDFS des services $S1$ et $S2$

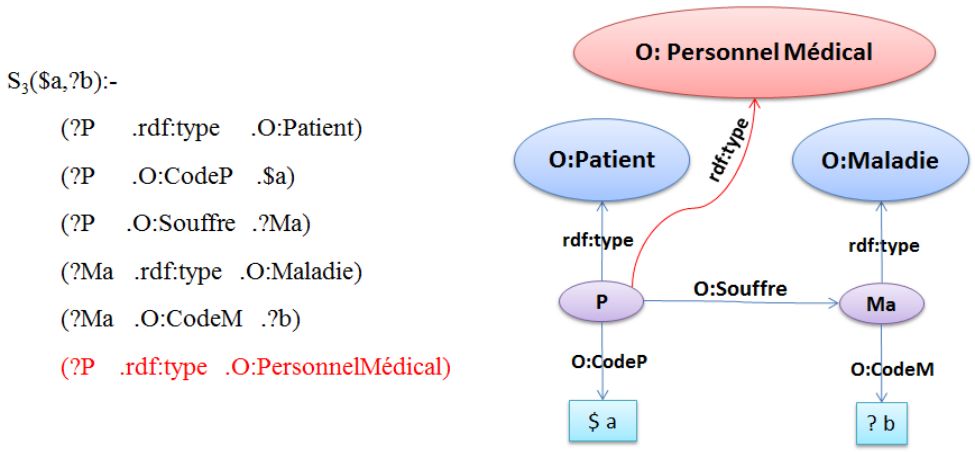

FIGURE 3.18 – Graphe RDFS du service $S3$

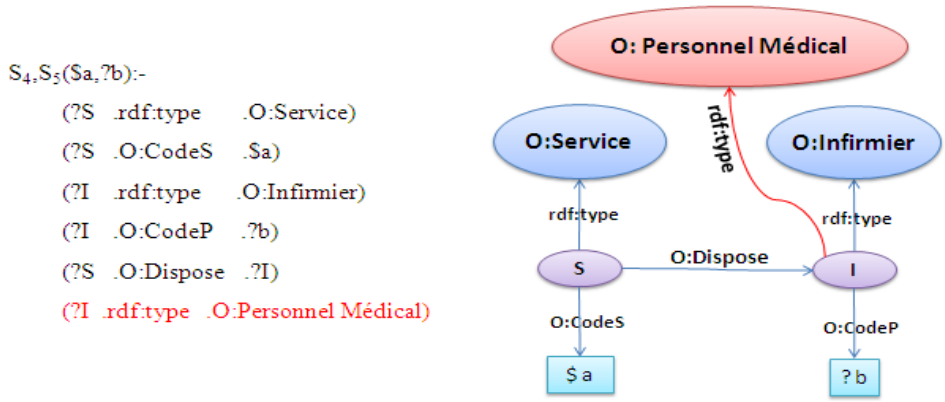

FIGURE 3.19 – Graphe RDFS des services $S4$ et $S5$

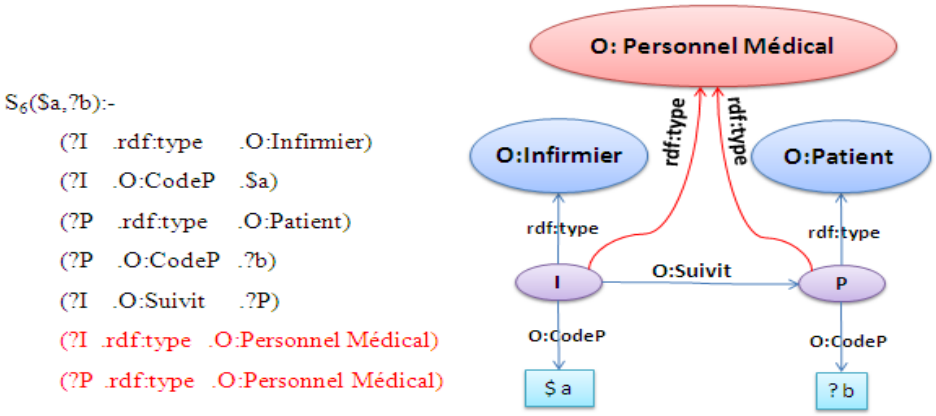

FIGURE 3.20 – Graphe RDFS du service $S6$

S_7($a,?b):-

(?P .rdf:type .O:Patient)

(?P .O:CodeP .$a)

(?R .rdf:type .O:Rapport)

(?R .O:CodeR .?b)

(?P .O:ConcernéPar .?R)

(?P .rdf:type .O:Personnel Médical)

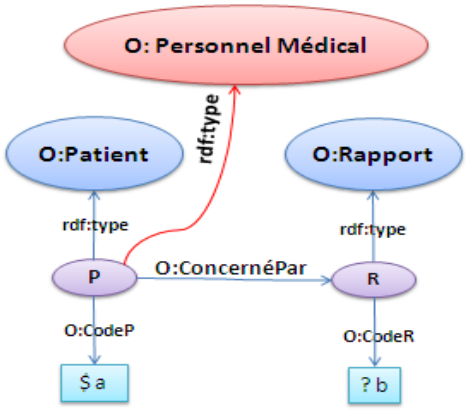

FIGURE 3.21 – Graphe RDFS du service $S7$

S_8($a,?b):-

(?P .rdf:type .O:Patient)

(?P .O:CodeP .$a)

(?M .rdf:type .O:Médecin)

(?M .O:CodeP .?b)

(?P .O:TraitéPar .?M)

(?P .rdf:type .O:Personnel Médical)

(?M .rdf:type .O:Personnel Médical)

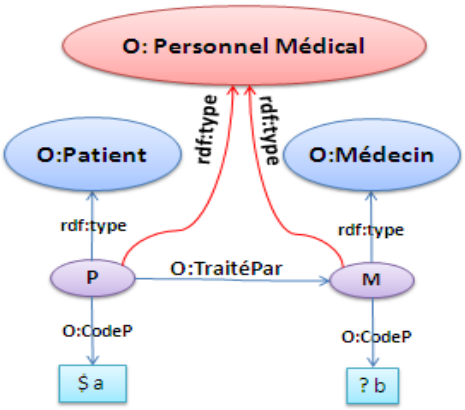

FIGURE 3.22 – Graphe RDFS des services $S8$

S_9($a,?b):-

(?S .rdf:type .O:Service)

(?S .O:CodeS .$a)

(?M .rdf:type .O:Médecin)

(?M .O:CodeP .?b)

(?S .O:Dispose .?M)

(?M .rdf:type .O:Personnel Médical)

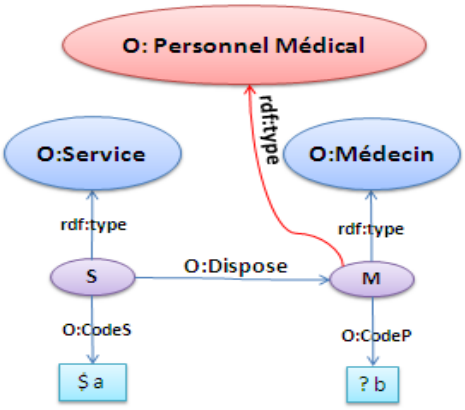

FIGURE 3.23 – Graphe RDFS des services $S9$

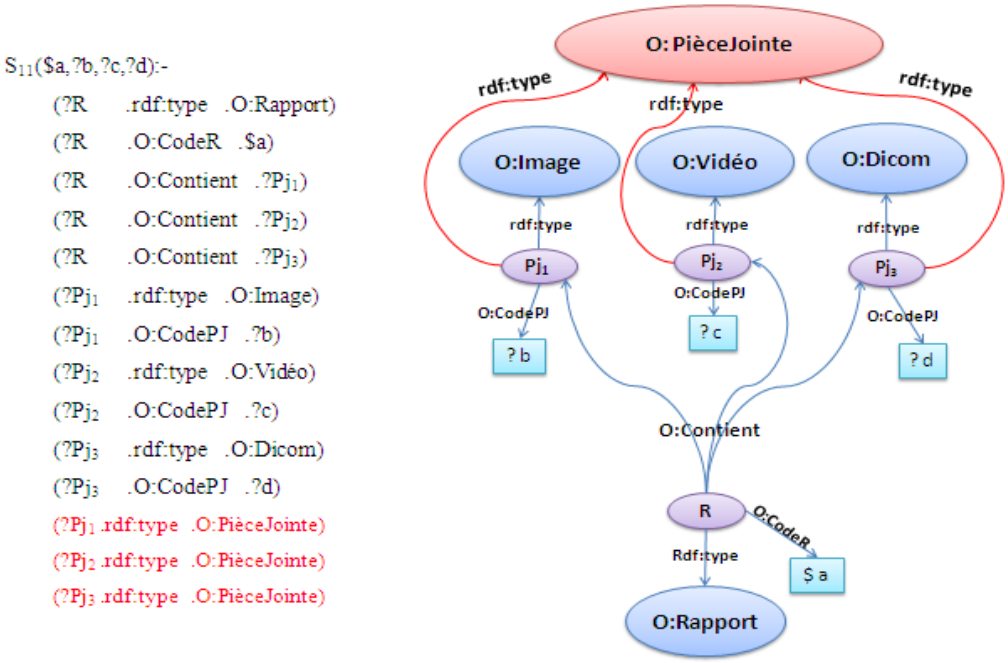

FIGURE 3.24 – Graphe RDFS du service $S11$

- **Association des fonctions Skolem aux noeuds classe**

Les variables indiquant les nœuds classe dans les RPVs sont associées une ***fonction skolem***. Une fonction skolem est définie comme suit : " A Skolem function returns a uniquely defined values for its arguments. Each of its invocation without arguments generates a new object. If it is invoked more than once for the same arguments it creates a new object only by the first invocation, by consecutive invocations it returns the identfier of the object created by the first evaluation" [13].

Cette fonction est utile pour fusionner les instances de classe issus de différents services. Dans le modèle de données RDF, une fonction Skolem associée à une classe RDF engendre une nouvelle "*valeur unique*", utilisée comme identifiant de l'instance de la classe lorsqu'elle est appelée avec quelques nouvelles valeurs de ses arguments, elle renvoie toujours la "même valeur" à chaque fois qu'elle est appelée avec les mêmes valeurs des arguments.

Nous avons utilisé les fonctions Skolem pour spécifier les DataType Properties qui peuvent être utilisés pour identifier de manière unique une instance de classe. Par exemple, la variable "?Ma" (de type Maladie) est associée à une fonction $F_1(CodeM)$ pour préciser que si deux instances ont le même CodeM alors ils désignent la même Maladie et donc peuvent être fusionnées. C'est un peu similaire à la contrainte de la clé primaire dans les bases de données. Les propriétés d'une fonction Skolem pour une classe particulière sont choisies par les experts du domaine. Les fonctions Skolem utilisées dans notre scénario sont : $F_1(CodeM)$,$F_2(CodeP)$,$F_3(CodeR)$,$F_4(CodeS)$ et $F_5(CodePJ)$.

3.6 Algorithme de réécriture de requête

Dans cette section, nous décrivons notre algorithme de réécriture de requête. Donnant une requête Q et un ensemble des services DaaS représentés par leurs vues correspondantes RPVs $V = V_1, V_2, \ldots, V_i$. L'algorithme de réécriture réécrit la requête Q comme une composition de services dont l'union de leurs graphes RDF (noté par G_V) couvre le graphe RDF de la requête (noté par G_Q). Le graphe G_V couvre le graphe G_Q ssi :

- Tous les Noeuds Classe dans G_Q sont dans G_V.
- Tous les Object Properties reliant les Noeuds Classe dans G_Q reliant aussi les Noeuds Classe correspondants dans G_V.
- Il y a un `containment mapping` β : $G_Q \rightarrow G_V$ telque :
  - β fait correspondre les Noeuds Classe dans G_Q à des Noeuds Classe dans G_V(i.e. ils ont le même type de classe).
  - β fait correspondre chaque Noeud littéral dans G_Q à un noeud littéral dans G_V.
  - Les variables distinguées dans Q sont fournis par composition de services.

La construction des compositions comprend deux phases : (a) trouver les sous-graphes de G_Q qui sont couverts par chaque RPV et (b) générer le service composite.

3.6.1 Trouver les sous graphes pertinents

L'algorithme de réécriture de requête compare la requête Q avec chaque vue V_i dans V et détermine les Noeuds Classe et les Object Properties de Q qui sont couverts dans chaque V_i et stocke ces informations dans une table appelée `table de couverture`. Dans cette étape, l'algorithme essaye de déterminer les vues pertinentes. L'algorithme de réécriture considère les deux cas suivants pour enrichir la table de couverture :

1. **Cas 1 (Noeuds Classe Couverts)** : On suppose que : $C_Q \in$ Q *et* $C_V \in V_i$ tel que C_Q et C_V ont le même type. On dit que RPV V_i couvre le noeud C_Q si les quatre conditions suivantes sont vérifiées :

   (a) Si C_Q a une variable distinguée x dans la requête Q (i.e. un DataType Property de C_Q est lié à une variable distinguée dans Q) alors le même DataType Property de C_V est projeté dans V_i (i.e. relié à une variable distinguée dans V_i) ou il peut être récupéré car tous les DataType Properties utilisés dans la fonction skolem de C_Q sont projetés dans V_i.

   (b) Si C_Q a une variable existentielle x dans la requête (i.e. un DataType Property de C_Q est lié à une variable existentielle x dans Q) alors une des conditions suivantes doit être vraie :

      i. La variable x correspond à une variable distinguée dans la vue V_i .

      ii. Le DataType Property lié à x peut être récupéré car tous les DataType Properties utilisés dans la fonction skolem de C_Q sont projetés dans V_i .

      iii. Tous les Noeuds Classe dans Q ayant le x dans leurs triplets sont couverts dans V_i et la jointure entre ces classes à travers x est appliquée dans V_i .

   (c) Si C_Q a une constante x dans ses triplets et x satisfait la contrainte posée sur le DataType Property que lui correspond dans V_i alors V_i projecte le DataType Property de C_V qui correspond à la constante x ou ce DataType peut être récupéré.

(d) Si C_Q est impliqué dans un Object Property P dans Q alors la vue V_i projecte les DataType Properties utilisés dans la fonction skolem de C_V ou elle doit couvrir P.

2. **Cas 2 (Object Properties Couverts)** : On suppose que P est un Object Property de la requête Q et P est inclue dans V_i tel que les Noeuds Classe liés par P dans V_i correspendent aux Noeuds Classe liés par P dans la requête Q (i.e. ont le même type). On dit que l'Object Property P est couvert par V_i SSi :

(a) V_i projecte les DataType Properties utilisés dans la fonction skolem de chaque Noeud Classe lié par P ; ou

(b) V_i couvre les Noeuds Classe liés par P que leurs DataType Properties utilisés dans ses fonctions skolem ne sont pas projetés dans V_i.

• **Exemple**

Laissez-nous maintenant illustrer les cas mentionnés ci-dessus en utilisant les services et la requête Q décrite dans notre scénario. Nous considérons les services S_1 et S_2 comme services candidats. Chacun de ces deux services retourne la liste des rapports fournis par un médecin ; mais S_1 est éliminé parce que le code du médecin youcef 'p100' est en dehors de la plage acceptée par S_1. S_2 a un Object Property 'fournit', les Nœuds Classe $S_2.M$ et $S_2.R$ liés par Object Property dans S_2 correspondent aux Nœuds Classe $Q.M$ et $Q.R$ liés par cette Object Property dans Q comme il est mentionné dans le Cas 2. L'Object Property 'fournit' est couvert par le service S_2 parce qu'il projecte les DataType Properties (CodeP, CodeR) utilisés dans la fonction skolem de chaque Nœud Classe lié par cette Object Property (i.e. les DataTypes Properties CodeP et CodeR correspondent respectivement aux variables distinguées '$a' et ' ?b' dans S_2). Le mapping β correspond est le suivant : $Q.M \rightarrow S_2.M$, Q.R$\rightarrow S_2.R$, ('p100')$\rightarrow\ a$, $x_6 \rightarrow\ b$

En plus, S_2 a un Nœud Classe $S_2.M$ qui correspond à $Q.M$, tous les DataTypes Properties de $Q.M$ qui sont liés à des variables distinguées dans Q sont aussi liés

à des variables distinguées dans S_2 cela satisfait la condition 'a' du cas 1. D'autre part, la condition 'b' est satisfaite du fait que $Q.M$ a des Datatype Properties qui sont liés à des variables existentielles dans la requête et que les deux sous conditions de la condition 'b' sont vraies. La condition 'c' est aussi satisfaite car $Q.M$ a une constante 'p100' qui satisfait la contrainte posée sur le Datatype Property qui lui correspond dans S_2 et que ce dernier projecte le Datatype Property de $S_2.M$ qui correpond à 'p100'. De même, la condition 'd' est aussi satisfaite puisque $Q.M$ est impliqué dans l'Object Property 'foutnit' dans la requête et que S_2 projecte le datatype property utilisé dans la fonction skolem de $S2.M$.

3. Présentation de l'algorithme 1

L'algorithme 1 qui permet de trouver les sous graphes RDF pertinents est divisé en deux parties : la première partie est consacrée pour la vérification du cas1 (lignes 4-19) et la deuxième partie pour la vérification du cas 2 (lignes 20-35) . Il a comme entrée une requête RDF Q et un ensemble de vues RDF et il génère comme sortie la table de couverture contenant les Noeuds Classe et les Object Properties Couverts par chaque service.

Dans la première partie, l'algorithme 1 compare chaque Noeud Classe C_Q dans Q avec chaque Noeud Classe C_V dans les vues V_i, et s'ils ont le même type il fait appel au sous algorithme 1 (NoeudClasseCouvert) dans la ligne 8 pour tester la couverture du Noeud Classe C_Q. Le sous algorithme 1 permet de vérifier les quatre conditions (a,b,c et d) citées précédemment dans le cas 1. Il vérifie la condition (a) (lignes 3-6), la condition (b) (lignes 16-39), la condition (c) (lignes 7-15) et la condition (d) (lignes 40-56). Si chacune de ces conditions est vérifiée, le sous algorithme 1 retourne la variable $TestCN$ avec la valeur vraie. A ce niveau l'algorithme 1 va créer une nouvelle ligne dans la table de couverture qui contient deux colonnes. Il va insérer dans la première colonne le service qui couvre le Noeud Classe C_Q et dans la deuxième colonne le Noeud Classe Couvert C_Q.

Dans la deuxième partie, l'algorithme 1 compare chaque Object Property P_Q dans Q avec chaque Object Property P_V dans les vues, s'ilsl ont le même domaine et le

même range, il fait appel au sous algorithme 2 (ObjectPropertyCouvert) dans la ligne 24 pour tester la couverture de l'Object Property P_Q. Le sous algorithme 2 va prendre au début les Noeuds Classe dans Q et dans V_i qui sont liés respectivement par P_Q et P_V (lignes 3-4). Si la condition (a) du cas 2 est vérifiée, il retourne la variable $TestOP$ avec la valeur vrai et par la suite, l'algorithme 1 va créer une nouvelle ligne dans la table de couverture et va insérer le service ainsi que l'Object Property couvert. Dans le cas où la condition (b) du cas 2 est vérifiée (lignes 8 et 20) , le sous algorithme 2 fait appel au sous algorithme 1 (lignes 9 et 21) pour la vérification de la couverture du Noeud Classe et si ce dernier est couvert le sous algorithme 2 va créer une nouvelle ligne dans la table de couverture et va insérer le service ainsi que le Noeud Classe couvert.

Algorithme 1 : Trouver les sous graphes RDF pertinents

Entrées : Q une requête RDF, V un ensemble des vues RDF
Sorties : une matrice T (table de couverture)

// * f : C $\times\{distinguished, existentielle, constante, skolem\} \longrightarrow$D
// Avec C: ensemble des Noeuds Classe, D: ensemble des DataType Properties
// La fonction $f(c)$ retourne soit:
// - Un ensemble des DataType Properties reliés à des variables distinguées.
// - Un ensemble des DataType Properties reliés à des variables existentielles.
// - Un ensemble des DataType Properties reliés à des constantes.
// - Un ensemble des DataType Properties utilisés dans les fonctions skolem.
// * La fonction $rdp(c)$ retourne les DataType Properties du concept c qui sont récupérés.
// * La fonction $cnt(c)$ retourne l'ensemble des constantes du concept c.
// * La fonction $tcn_x(d)$ retourne les Noeuds Classe de x qui ont le DataType Property d dans leur triplets.
// * $Pred$ c'est la contrainte posée sur le DataType Property relié à la constante.
// * $const$ c'est la valeur constante d'un DataType Property.
// * t c'est l'ensemble des Noeuds Classe couverts.

```
1  begin
2      i ←0
3      t ←∅
4      foreach Noeud Classe C_Q dans Q do
5          foreach Vue V_i dans V do
6              foreach Noeud Classe C_V dans V_i do
7                  if TypeClasse(C_Q) = TypeClasse(C_V) then
8                      if NoeudClasseCouvert(C_Q, C_V) then
9                          begin
10                             i ← i + 1
11                             T[i, 1] ← V_i
12                             T[i, 2] ← C_Q
13                             t ← t ∪ {C_Q}
14                         end
15                     end if
16                 end if
17             end foreach
18         end foreach
19     end foreach
20     foreach Object Property P_Q dans Q do
21         foreach Vue V_i dans V do
22             foreach Object Property P_V dans V_i do
23                 if (TypeDomain(P_Q) = TypeDomain(P_V)) and (TypeRange(P_Q) =
                   TypeRange(P_V)) then
24                     if ObjectPropertyCouvert(P_Q, P_V) then
25                         begin
26                             i ← i + 1
27                             T[i, 1] ← V_i
28                             T[i, 2] ← C_Q
29                             t ← t ∪ {P_Q}
30                         end
31                     end if
32                 end if
33             end foreach
34         end foreach
35     end foreach
36 end
```

Sous Algorithme 1 : NoeudClasseCouvert(C_Q, C_V)

Entrées : C_Q, C_V deux Noeuds Classe
Sorties : $TestCN$ une variable booléene

```
// La fonction contrainte(const) retourne vrai s'il existe une contrainte sur le
   DataType Property relié à la constante const//
1  begin
2      TestCN ← False
3      TestA ← False
4      if f(C_Q, distinguished) ⊂ (f(C_V, distinguished) ∪ rdp(C_V)) then
5          TestA ← True
6      end if
7      TestC ← False
8      if f(C_Q, constante) ⊂ (f(C_V, distinguished) ∪ rdp(C_V)) then
9          TestC ← True
10     end if
11     foreach const ∈ cnt(C_Q) do
12         if contrainte(const) and (const ∉ Pred) then
13             TestC ← False
14         end if
15     end foreach
16     TestB ← False
17     foreach DataType Property d dans f(C_Q, existentielle) do
18         if d ∈ (f(C_v, distinguished) ∪ rdp(C_V)) then
19             TestB ← True
20         else
21             foreach C_K ∈ tcn_Q(d) do
22                 if (C_K map to C_m dans V_i) and (C_m ∈ tcn_Vi(d)) then
23                     if C_K ∉ t then
24                         if NoeudClasseCouvert(C_K, C_m) then
25                             begin
26                                 i ← i + 1
27                                 T[i, 1] ← V_i
28                                 T[i, 2] ← C_K
29                                 t ← t ∪ {C_K}
30                             end
31                         end if
32                     end if
33                 end if
34             end foreach
35             if tcn_Q(d) ⊆ t then
36                 TestB ← True
37             end if
38         end if
39     end foreach
40     TestD ← False
41     if C_Q est impliqué dans un Object Property p dans Q then
42         if f(C_V, skolem) ⊂ f(C_V, distinguished) then
43             TestD ← True
44         else
45             if p correspond p_V dans V_i then
46                 if ObjectPropertyCouvert(p, p_V) then
47                     begin
48                         TestD ← True
49                         i ← i + 1
50                         T[i, 1] ← V_i
51                         T[i, 2] ← p
52                     end
53                 end if
54             end if
55         end if
56     end if
57     TestCN ← (TestA and TestB and TestC and TestD)
58 end
```

Sous Algorithme 2 : ObjectProperyCouvert(P_Q, P_V)

Entrées : P_Q, P_V deux Object Properties
Sorties : $TestOP$ une variable booléene

```
1  begin
2  |  TestOp ← True
3  |  Prendre les noeuds (C_Q, C_Q1) lies par P_Q dans Q
4  |  Prendre les noeuds (C_V, C_V1) lies par P_V dans V_i
5  |  if (TypeClasse(C_Q) ≠ TypeClasse(C_V)) or (TypeClasse(C_Q1) ≠ TypeClasse(C_V1))
   |  then
6  |  |  TestOP ← False
7  |  else
8  |  |  if f(C_Q, skolem) ⊄ f(C_V, distinguished) then
9  |  |  |  if NoeudClasseCouvert(C_Q, C_V) then
10 |  |  |  |  begin
11 |  |  |  |  |  i ← i + 1
12 |  |  |  |  |  T[i, 1] ← V_i
13 |  |  |  |  |  T[i, 2] ← C_Q
14 |  |  |  |  |  t ← t ∪ {C_Q}
15 |  |  |  |  end
16 |  |  |  else
17 |  |  |  |  TestOp ← False
18 |  |  |  end if
19 |  |  end if
20 |  |  if f(C_Q1, skolem) ⊄ f(C_V1, distinguished) then
21 |  |  |  if NoeudClasseCouvert(C_Q1, C_V1) then
22 |  |  |  |  begin
23 |  |  |  |  |  i ← i + 1
24 |  |  |  |  |  T[i, 1] ← V_i
25 |  |  |  |  |  T[i, 2] ← C_Q1
26 |  |  |  |  |  t ← t ∪ {C_Q1}
27 |  |  |  |  end
28 |  |  |  else
29 |  |  |  |  TestOp ← False
30 |  |  |  end if
31 |  |  end if
32 |  end if
33 end
```

Service	**Noeud Classe/Object Property couvert**
S_1	Q.R, Fournit
S_2	Q.M, Q.R, Fournit
S_3	Q.P, Q.Ma, Souffre
S_4	Q.I, Dispose
S_5	Q.I, Q.S, Dispose
S_6	Q.I, Q.P, Suivit
S_7	Q.R, Q.P, ConcernéPar
S_8	Q.M, Q.P
S_9	Q.M, Q.S, Dispose
S_{10}	Q.R
S_{11}	Q.R

TABLE 3.2 – Table de couverture

3.6.2 Génération du service composite

Après la génération de la table de couverture dans l'étape précédente nous passons à la deuxième étape de l'algorithme qui est la génération du service composite. Dans cette étape, l'algorithme explore les différentes combinaisons à partir de la table de couverture pour couvrir le graphe de la requête. Il considère la combinaison des ensembles disjoints des Noeuds Classe et des Object Properties couverts. Une combinaison est dite valide si les deux conditions suivantes sont vérifiées :

- Elle couvre l'ensemble des Nœuds Classe et des Object Properties dans Q (condition vérifiée par l'algorithme 2).
- Elle est exécutable : une composition est dite exécutable si tous les paramètres d'entrée nécessaire pour l'invocation de ses services sont liés (i.e. ils sont connus) ou peuvent être liés par l'invocation des services dont les paramètres d'entrées sont liés (condition vérifiée par l'algorithme 3).

• **Génération des compositions candidates**

L'algorithme 2 représenté ci-dessous nous permet de générer les différentes compositions condidates qui couvrent le graphe de la requête Q. Il reçoit comme entrée la table de couverture, il associe pour chaque Nœud Classe ou Object Property de la requête Q un Subset Set_i contenant les services couvrant le nœud ou l'object property de la requête (lignes 2-11). Ensuite, à partir de chaque Subset $Set_i \ldots Set_n$, il combine les services (lignes 13). Pour chaque combinaison C, il élimine la redondance des services (ligne 15-20) et après, il vérifie la couverture de la requête par cette combinaison et la disjonction entre les services appartenant à la combinaison (lignes 21-32).

Algorithme 2 : Génération des compositions candidates

Entrées : La table de couverture T
Sorties : $EnsComp$ ensemble des compositions candidates

```
// * La fonction Nop(S) retourne les Noeuds Classe et les Object Properties
   couverts par le service S.
// * C est une composition des services.
// * CNOP peut être un Noeud Classe ou un Object Property.
```

```
1  begin
2      i ← 0
3      foreach CNOP in Q do
4          i ← i+1
5          Set_i ← ∅
6          for j ← 1 to length(T) do
7              if CNOP = T[j,2] then
8                  Set_i ← Set_i ∪ {T[j,1]}
9              end if
10         end for
11     end foreach
12     EnsComp ← ∅
13     From every Subset Set_i.....Set_n  Combinez les services
14     begin
15         C' ← {C[1]}
16         for j ← 2 to length(C) do
17             if C[j] ∉ C' then
18                 C' ← C' ∪ {C[j]}
19             end if
20         end for
21         UnionNop ← Nop(C[1])
22         InterNop ← Nop(C[1])
23         for h ← 2 to length(C) do
24             begin
25                 UnionNop ← UnionNop ∪ Nop(C[h])
26                 InterNop ← InterNop ∩ Nop(C[h])
27             end
28         end for
29         if (UnionNop = Nop(Q)) and (InterNop = ∅) then
30             EnsComp ← EnsComp ∪ {C}
31         end if
32     end
33 end
```

La table 3.3 représente quelques exemples de compositions candidates.

Combinaisons	**Services**
C_1	$S_2, S_1, S_3, S_4, S_5, S_7, S_6$
C_2	S_2, S_5, S_6, S_3, S_7
C_3	$S_9, S_{11}, S_8, S_4, S_3, S_1, S_7, S_6$

TABLE 3.3 – Exemples de compositions candidates

• **Vérification d'exécutabilité des compositions candidates**

L'algorithme 3 représenté ci-dessous permet de vérifier l'exécutabilité des compositions candidates, il reçoit comme entrée l'ensemble de compositions candidates obtenues dans l'algorithme 2 et retourne comme sortie un ensemble de compositions exécutables. L'algorithme vérifie l'exécutabilité pour chaque composition C dans l'ensemble des compositions candidates (lignes 3-20). Il commence par l'initialisation de l'ensemble A des services invoqués (ligne 4). Ensuite, il initialise la variable booléenne "$invoque$" à $False$ et il parcoure la composition service par service, si le service concerné n'appartient pas à l'ensemble des services invoqués et que les entrées de ce service sont inclues dans l'ensemble B (ensemble des variables liées à des constantes dans la requête Q) dans ce cas là, il ajoute ce service à l'ensemble des services invoqués, ses sorties à l'ensemble B et il affecte la valeur $True$ à la variable "$invoque$" (ligne 6-13), ces étapes sont répétées jusqu'à ce que la variable "$invoque$" devient $Fausse$, ce qui signifie que tous les services de la composition C sont invoqués ou bien, il n' y a plus de service à invoquer (la composition n'est pas exécutable). Enfin,il vérifie si l'ensemble A est équivalent à la composition C, dans ce cas là, il ajoute A à l'ensemble des compositions exécutables (lignes 17-19).

Algorithme 3 : Vérification d'exécutabilité des compositions candidates

Entrées : *EnsComp* ensemble des compositions condidates
Sorties : *EnsExe* ensemble des compositions exécutables

```
// * A ensemble des services invoqués.
// * B ensemble des variables liées à des constantes dans la requête Q.
// * input(S) une fonction qui retourne l'ensemble des variables d'entrée d'un
   service.
// * output(S) une fonction qui retourne l'ensemble des variables de sortie d'un
   service.
1  begin
2      EnsExe ← ∅
3      foreach Composition C dans EnsComp do
4          A ← ∅
5          repeat
6              invoque ← False
7              for i ← 1 to lenght(C) do
8                  if (C[i] ∉ A) and (input(C[i]) ⊆ B) then
9                      begin
10                         A ← A ∪ {C[i]}
11                         B ← B ∪ output(C[i])
12                         invoque ← True
13                     end
14                 end if
15             end for
16         until invoque = False
17         if A ≡ C then
18             EnsExe ← EnsExe ∪ {A}
19         end if
20     end foreach
21 end
```

La vérification de l'exécutabilité de la composition C_2 est représentée dans la table 3.4

Variables liées	**Services invoqués**
	$S_2('p100', ?x_6), S_5('S12', ?x_3)$
x_3, x_6	$S_2('p100', ?x_6), S_5('S12', ?x_3), S_6(\$x_3, ?x_2)$
x_3, x_6, x_2	$S_2('p100', ?x_6), S_5('S12', ?x_3), S_6(\$x_3, ?x_2),$ $S_3(\$x_2, ?x_5), S_7(\$x_2, ?x_6)$
x_3, x_6, x_2, x_5	Tous les services

TABLE 3.4 – Vérification de l'exécutabilité de la composition C_2

3.6.3 Exécution du service composite

Un service composite doit être exécuté dans un ordre particulier en fonction de son modèle d'accès. Si un service S_j a une entrée x qui est obtenue à partir d'une sortie y de S_i alors S_j doit être précédé par S_i dans le plan d'exécution. Dans ce cas S_i est appelé le `Parent` du S_j. Nous définissons un graphe de dépendance comme un graphe orienté acyclique G dans lequel les nœuds correspondent à des services et les arcs correspondent à des contraintes de dépendances entre les services. La figure suivante représente le graphe de dépendance pour la composition C_2. Il y a une dépendance entre $S_5(\$x_4, ?x_3)$ et $S_6(\$x_3, ?x_2)$ pour cela, il faut que $S_5(\$x_4, ?x_3)$ doit être exécuté en premier. $S_5(\$x_4, ?x_3)$ est appelé le parent du service $S_6(\$x_3, ?x_2)$. Il y a aussi une dépendance entre $S_6(\$x_3, ?x_2)$ et $S_3(\$x_2, ?x_5)$ et entre $S_6(\$x_3, ?x_2)$ et $S_7(\$x_2, ?x_6)$, donc, $S_6(\$x_3, ?x_2)$ est exécuté avant $S_3(\$x_2, ?x_5)$ et $S_7(\$x_2, ?x_6)$ et ces derniers peuvent être exécutés en parallèle.

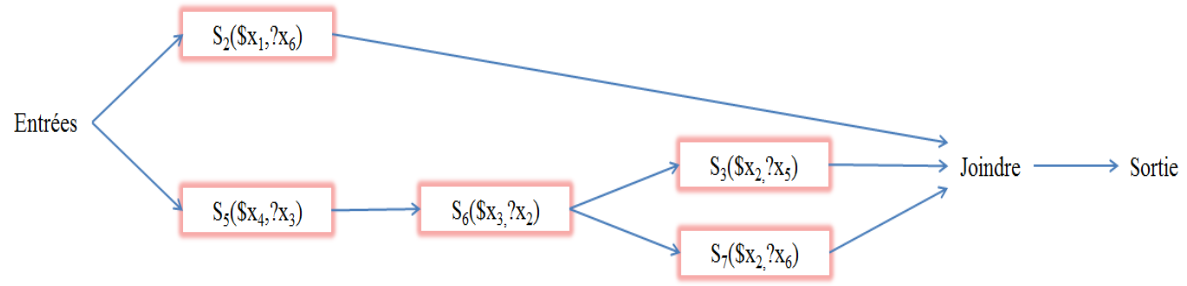

FIGURE 3.25 – Exemple de graphe de dépendance

L'algorithme 4 cité ci-dessous présente la façon dont les services composites sont exécutés par le système WSMS (Web Service Management System). L'algorithme a comme entrée l'ensemble de compositions exécutables et il retourne comme sortie le résultat de la requête Q. Au début, il crée un thread T_i pour chaque service S_i dans une composition C_i. Le thread T_i prend ses tuples d'entrée à partir d'un thread joignant séparé J_i qui joint les sorties des parents du S_i. Si S_i n'a pas de parents dans C_i, alors T_i prend ses entrées à partir de la relation d'entrée I qui contient toutes les valeurs spécifiques dans Q (i.e. les constantes de la requête). Le thread T_i invoque S_i pour chaque tuple d'entrée, filtre les tuples retournés, les joindre avec les tuples d'entrée et les écrits à son sortie. Le résultat de la requête est obtenu à partir de la sortie du thread joignant J_{out} qui relie les sorties de tous les services qui sont des feuilles dans le graphe de dépendance de C_i.

Algorithme 4 : Exécution du service composite

Entrées : $EnsExe$ ensemble de compositions exécutables
Sorties : Résultat de la requête Q

```
// * I ensemble contenant toutes les valeurs spécifiques dans Q (i.e.les
   constantes de la requête)
// *Thread T_i : Thread d'invocation. Ce thread invoque un Service Web avec les
   sorties de ses parents.
// *Thread J_i : Thread joignant. Il permet de joindre les sorties des services
   parents du S_i dans la composition C_i.
// *Thread J_out : Thread de sortie. Il obtient la sortie de la composition en
   joignant les sorties des services situant dans la composition.
1  begin
2      foreach Service Web S_i dans C_i do
3          Créer un thread T_i
4          if S_i n'a pas de parents dans C_i then
5              T_i prend son entrée depuis la relation d'entrée I
6          else
7              if S_i a un seul parent S_p dans C_i then
8                  T_i prend son entrée depuis la sortie de T_p
9              else
10                 begin
11                     Créer un thread joignant J_i
12                     T_i prend son entrée depuis la sortie de J_i
13                 end
14             end if
15         end if
16     end foreach
17     Créer un thread joignant J_out comme résultat de la requête
18 end
```

3.7 Conclusion

Nous avons présenté dans ce chapitre l'architecture globale de notre système de médiation permettant d'interroger et de composer automatiquement les Services Web DaaS afin de répondre aux requêtes des utilisateurs. Les services DaaS sont décrits par des vues RDF à partir d'une ontologie de médiation. Ces vues sont enrichies par les contraintes sémantiques RDFS. Nous avons présenté également une approche pour la composition des Services Web DaaS basée sur la réécriture d'une requête RDF. L'approche réécrit la requête directement en termes des Services Web disponibles. Toutefois, cela peut ne pas être toujours souhaitable, car le problème de réécriture de requêtes a en général une très grande complexité de l'ordre NP-complet [3] du fait qu'il peut impliquer la recherche grâce à un nombre exponentiel de réécritures, et cela peut présenter un problème d'évolutivité important pour l'algorithme de composition lorsque le nombre de Services Web DaaS disponibles est particulièrement important.

Conclusion Générale

Les Services Web sont de plus en plus adopté comme une méthode fiable, bien documentée et interopérables pour l'échange de données entre les entreprises modernes. La composition des Services Web DaaS permet de résoudre les requêtes des utilisateurs qui ne peuvent être résolues par des services individuels. Cependant, les techniques traditionnelles de composition des Services Web sont inapplicables aux Services Web DaaS.

Dans ce mémoire, nous avons présenté un algorithme de réécriture de requêtes en termes de vues permettant la composition des Services Web DaaS. L'algorithme est inspiré de l'algorithme MiniCon [2] adapté et appliqué à la réécriture de requête en termes de vues RDF accessibles via Services Web DaaS. Notre proposition traite ainsi la composition des Services Web DaaS comme réécriture de requête. Dans ce contexte, d'autres travaux ont traité cette thématique, notamment la thèse de Mahmoud Barhamgi [6]. Cependant, contrairement à notre travail, l'algorithme de réécriture de Mahmoud ne fait aucune référence aux algorithmes classiques de réécriture de requêtes. Également, notre approche est d'un point de vue algorithmique plus performante bien que nous n'avons pas fait d'étude approfondi pour prouver et démontrer cet assomption. Enfin, notre approche traite sans ambiguïté tous les cas possibles contrairement à l'approche de Barhamgi [6] qui laisse dans un certains nombre de cas certaines des questions sans réponses. Là aussi, nous n'avons pas eu le temps de prouver et démontrer cet état de fait.

Les perspectives que nous envisageons de conduire sont les suivantes :

– Améliorer le choix de la composition exécutable pour répondre à la requête d'utilisateur.

– Faire une étude approfondi pour prouver et démontrer que notre approche est plus performante du point de vue algorithmique et qu'elle traite tous les cas possibles contrairement à l'approche de Barhamgi.

– Extentier notre système en ajoutant la partie indexation afin qu'il devient un système d'indexation et d'interrogation complet.

Bibliographie

[1] B. J. Ailas Lahcene – *Système de recherche sémantique des images médicales-rsim-*, mémoire de fin d'études pour l'obtention du diplôme d'ingénieur d'état en informatique option "système d'information avancé", Université Abou Bakr Belkaid -Tlemcen-, 2010.

[2] J. J. Alon Y Halevy, Anand Rajaraman – « Querying heterogeneous information sources using source descriptions », *In Proc of VLDB* (1996), p. 251–262.

[3] Y. S. D. S. Alon Y Halevy, Alberto O Mendelzon – « Answering queries using views », *in PODS* (1995), p. 95–104.

[4] S. G. Antonio Bucchiarone – « A survey on services composition languages and models », *In Antonia Bertolino and Andrea Polini, editors, in Proceedings of International Workshop on Web Services Modeling and Testing (WS-MaTe2006)* (2006), p. 51–63.

[5] N. Arenaza – *Composition semi-automatique de services web*, Projet de master, Ecole Polytechnique Fédérale de Lausane, Février 2006.

[6] M. Barhamgi – « Composing daas web services application to ehealth », Thèse de doctorat, Université Claude Bernard Lyon I, 2010.

[7] A. BOUKHADRA – « La composition dynamique des services web sémantiques a base d'alignement des ontologies owl-s », Mémoire pour obtenir le diplôme de magsiter en informatique option système d'information et de connaissance « sic », Ecole national Supérieur d'Informatique, 2011.

[8] M. Carey – « Data delivery in a serviceoriented world : The bea aqualogic data services platform », *in SIGMOD Conference* (2006), p. 695–705.

[9] I. Composite Software – « http ://www.compositesw.com/products/, the composite data virtualization platform, août 2011 ».

[10] M. Corporation – « http ://www.jwc3.net/2008/06/project-astoria-aka-adonet-data.html, ado.net data services (”project astoria”),2008 ».

[11] Y. N. Gilpin Mike – « Information-as-a-service : What's behind this hot new trend ? », Tech. report, Forrester Research, 2007.

[12] T. Mellити – « Interropérabilité des services web complexes », Thèse de doctorat, Université Paris IX Dauphine, 2004.

[13] T. Pankowski – « Xml schema mappings using schema constraints and skolem functions », *in Knowledge-Driven Computing :Springer Berlin* (2008), p. 22.

[14] J. Ponge – « Model based analysis of time-aware web services interactions », Thèse de doctorat, Université Blaise Pascal - Clermont-Ferrand II, 2008.

[15] F. Rastier – « Ontologies », *Revue des sciences et technologies de l'information* (2004), p. 15–40.

[16] I. Red Hat – « Metamatrix enterprise data services platform, http ://www.redhat.com/jboss/platforms/dataservices/, juillet 2011 », 2007.

[17] T. R.Gruber – « A translation approach to portable ontology specifications », Tech. report, Knowledge Acquisition, 1993.

[18] W. S. Schahram Dustdar – « A survey on web services composition », *Int. J. Web and Grid Services* (2005).

[19] O. L. Tim Berners-Lee, James Hendler – « The semantic web », *Scientific American* (2001), p. 3.

[20] W3C – « http : //www.w3.org/tr/rdf-schema/,rdf schéma, juillet 2011 ».

[21] — , « http ://www.w3.org/rdf/,rdf, juillet 2011 ».

[22] — , « http ://www.w3.org/tr/rdf-sparql-query/, langage sparql, juillet 2011 ».

[23] — , « http ://www.w3.org/tr/ws-desc-reqs/, services web, juillet 2011 ».

[24] Xcalia – « Xcalia intermediation platform, http ://club-java.com/slides/23062005/02-samson-xcaliaintermediationplatform.ppt, août 2011 ».