

Chapitre 3

Conception de notre système de médiation

Sommaire

3.1	Introduction	29
3.2	Architecture globale du système	30
3.3	Exemple de motivation	31
3.4	Services DaaS et modèle de requête	35
3.4.1	Ontologie de médiation	35
3.4.2	Requête	36
3.4.3	Vue RDF Paramétrée (RPV)	38
3.5	Pré-traitement des vues RDF	42
3.6	Algorithme de réécriture de requête	47
3.6.1	Trouver les sous graphes pertinents	48
3.6.2	Génération du service composite	55
3.6.3	Exécution du service composite	59
3.7	Conclusion	60

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 permettant 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'appuie 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 permet 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 (Querying 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.

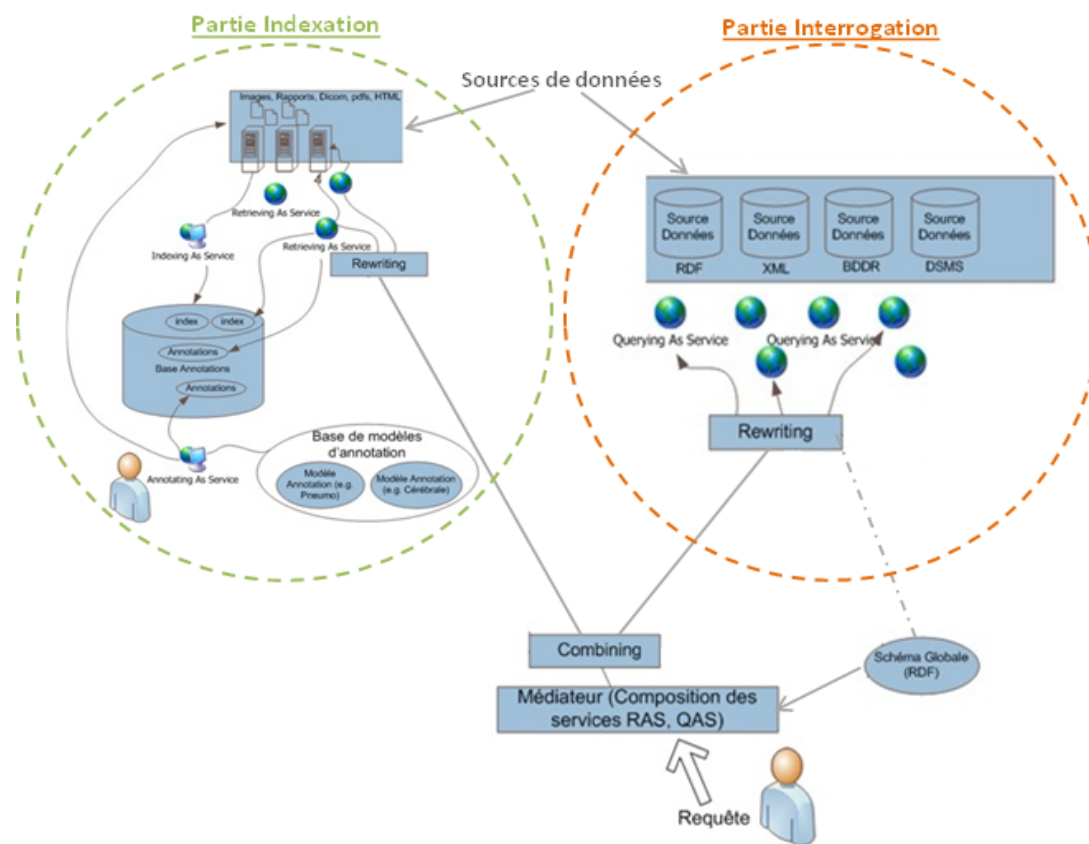


FIGURE 3.1 – Architecture globale du système

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)	$a \geq p150$
$S_2(\$a, ?b)$	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)	$a \leq S18$
$S_6(\$a, ?b)$	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 ' $a \leq S18$ ' 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.

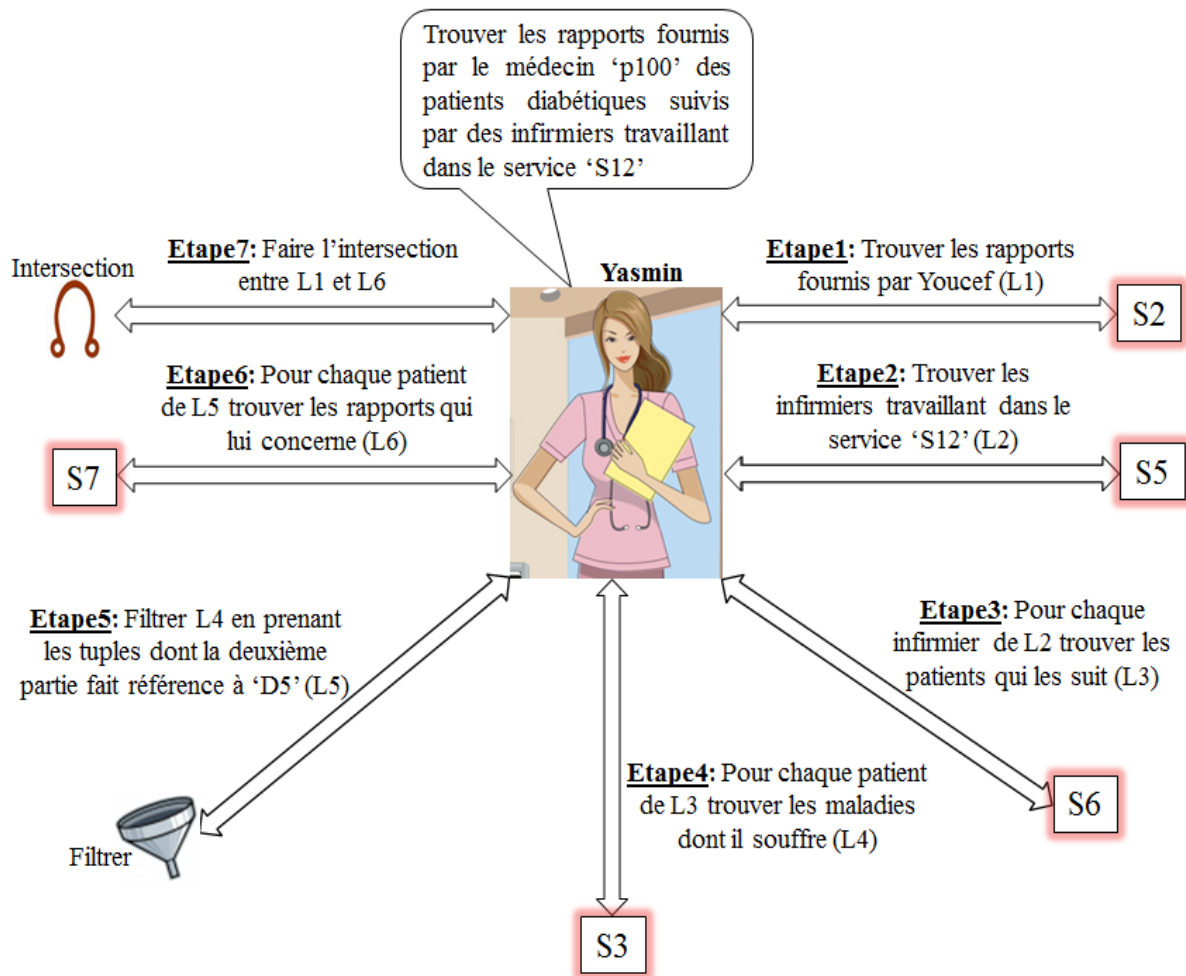


FIGURE 3.2 – Scénario de motivation proposé

- 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. Troisièmement, 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.

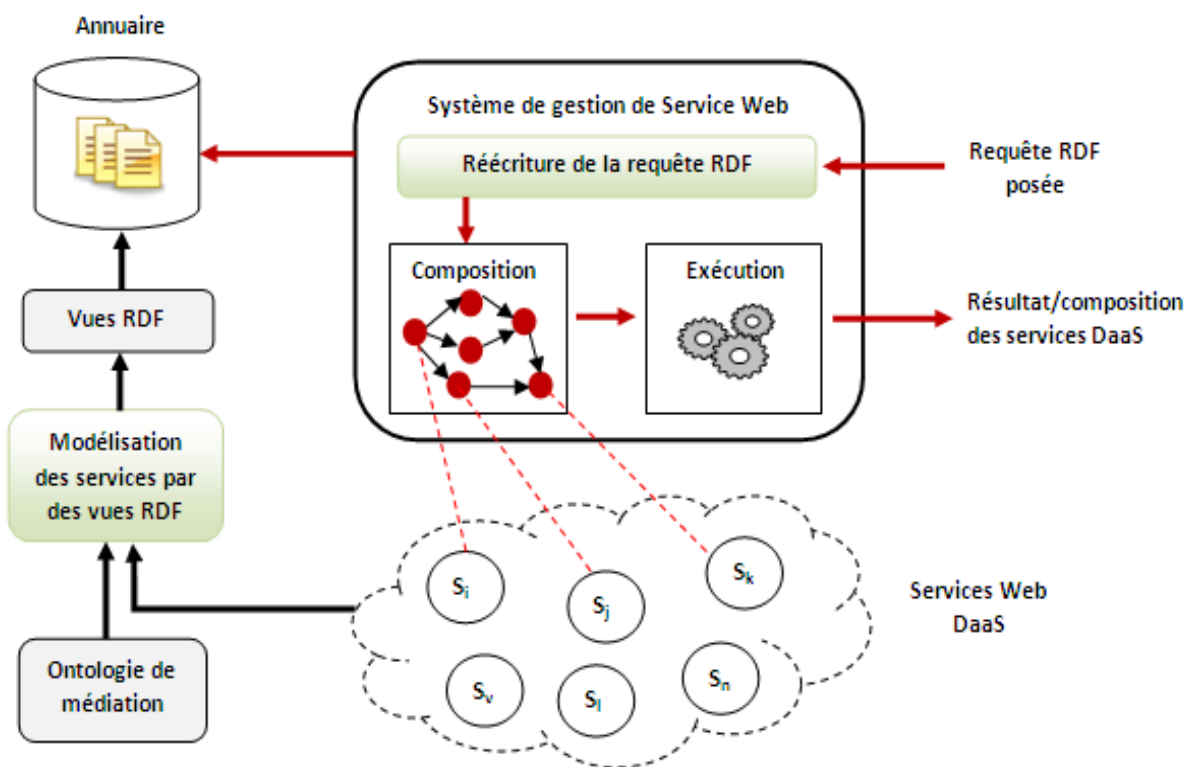


FIGURE 3.3 – Aperçu de l'approche proposée

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 $(C \times C)$, représentant les relations **sub-class of** entre les classes. Par exemple, $C_2 \text{ SC } C_1$ signifie que C_2 est une sous classe de C_1 .
- **SP** est une relation dans $[(OP \times OP) \cup (DP \times DP)]$. Elle représente les relations **sub-property of** entre les Properties dans OP ou dans DP. Par exemple, $p_2 \text{ SP } p_1$ signifie que p_2 est une sous propriété de p_1 .

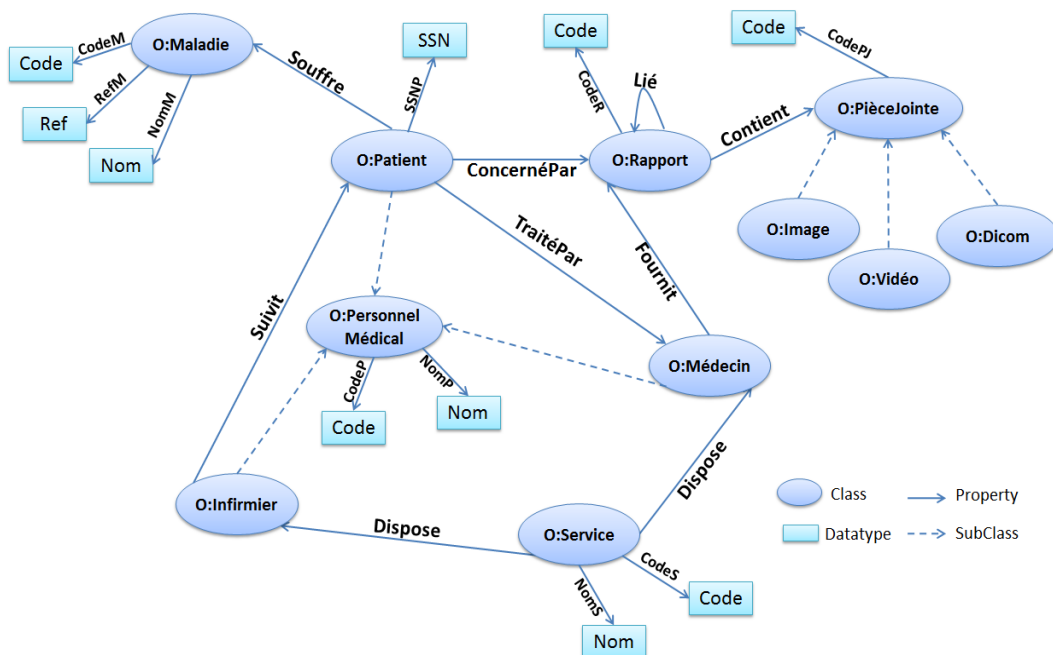


FIGURE 3.4 – Ontologie de médiation proposée

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 :

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

- $Q(\bar{X})$ est appelé l'entête de la requête, elle a la forme d'un prédicat relationnel.
- \bar{X} sont appelées les variables de l'entête ou les **variables distinguées**.
- \bar{Y} sont appelées les **variables existentielles**.
- $G(\bar{X}, \bar{Y})$ 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 : $x \Theta \text{ CONSTANT}$, 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' " .

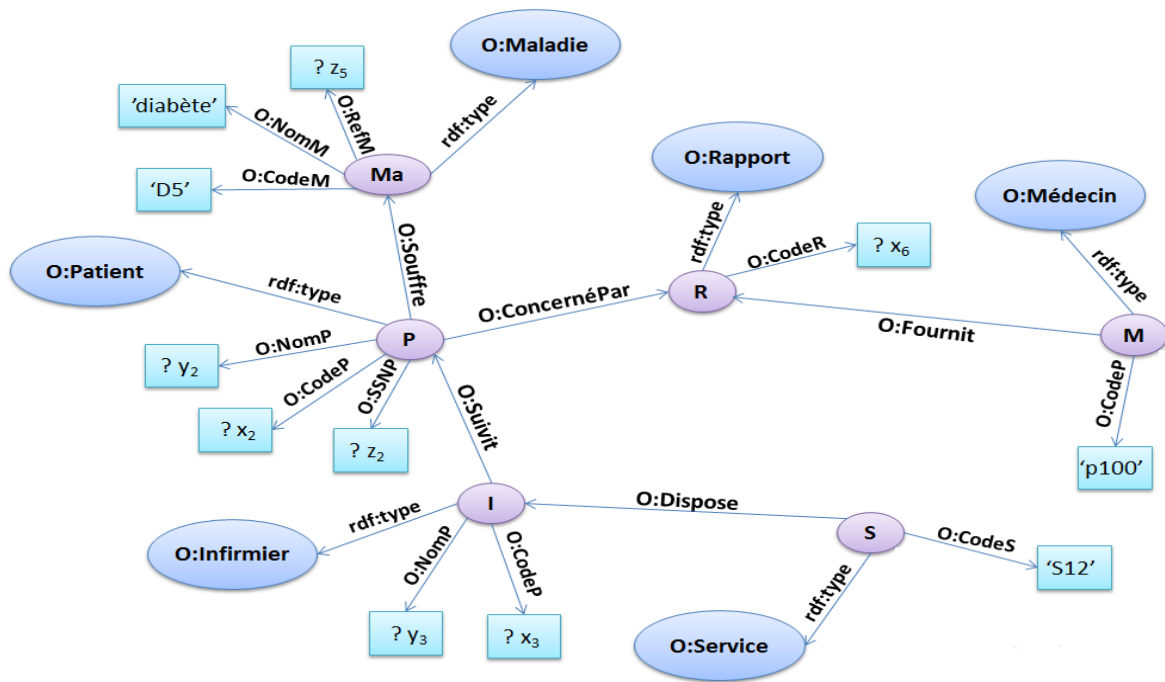


FIGURE 3.5 – Représentation graphique de la requête Q

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

?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 existentielles 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 RPVs⁹ 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

$$S_i(\bar{X}_i, \bar{Y}_i) : - \langle \Phi(\bar{X}_i, \bar{Y}_i, \bar{Z}_i), Ct_i \rangle, \text{ où :}$$

- \bar{X}_i est l'ensemble des variables indiquant les paramètres d'entrée nécessaires pour invoquer S_i , elles sont appelées *les variables d'entrée*.
- \bar{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**.
- $\Phi(\bar{X}_i, \bar{Y}_i, \bar{Z}_i)$ représente la relation sémantique entre les variables d'entrée et de sortie. \bar{Z}_i est l'ensemble des **variables existentielles** reliant \bar{X}_i et \bar{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 \bar{X}_i, \bar{Y}_i ou \bar{Z}_i . Une contrainte a la forme : $\Theta \text{ CONSTANT}$, 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 symboles "\$" et "?".

Les services S_1 et S_2 donnent les rapports (b) fournis par un médecin (a).

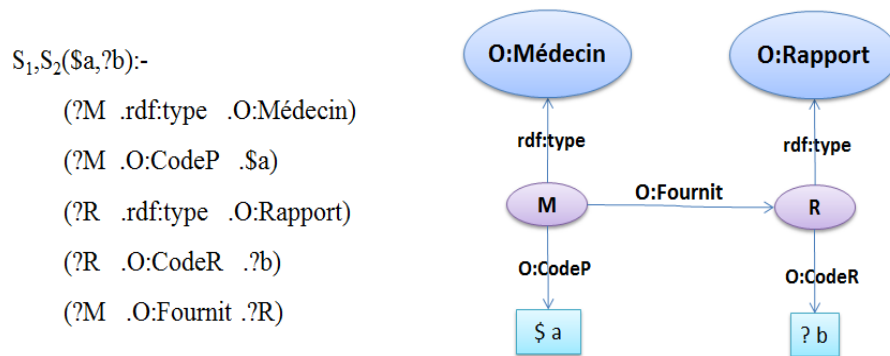


FIGURE 3.7 – Graphe RDF des services S_1 et S_2

Le service S_3 donne les patients (b) souffrant d'une maladie (a).

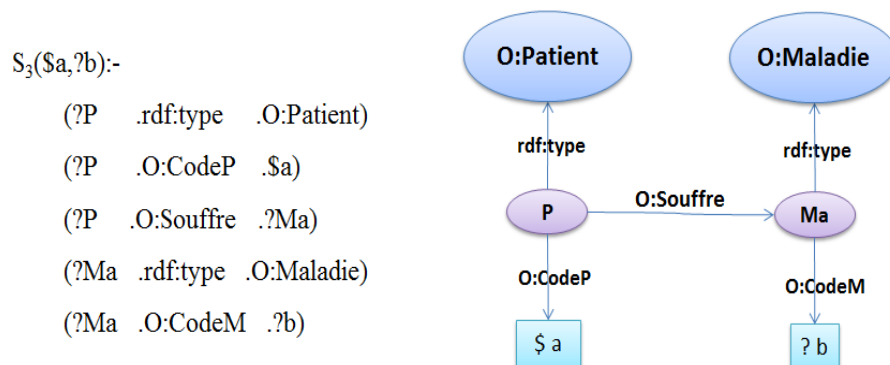


FIGURE 3.8 – Graphe RDF du service S_3

Les services S_4 et S_5 donnent les infirmiers (b) travaillant dans un service (a).

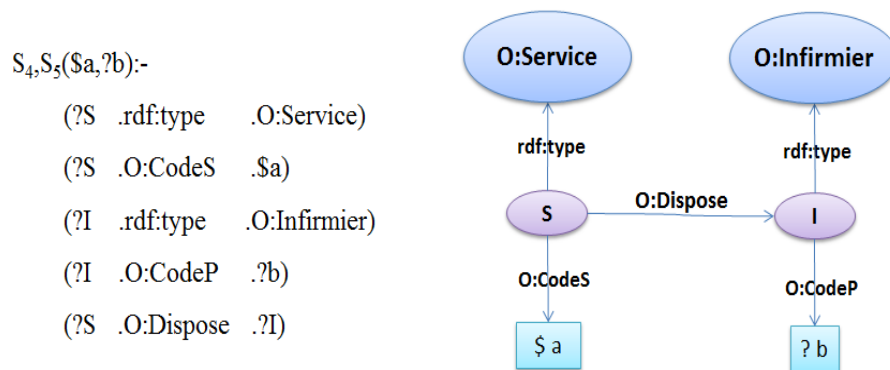


FIGURE 3.9 – Graphe RDF des services S_4 et S_5

Le service S_6 donne les patients (b) suivi par un infirmier (a).

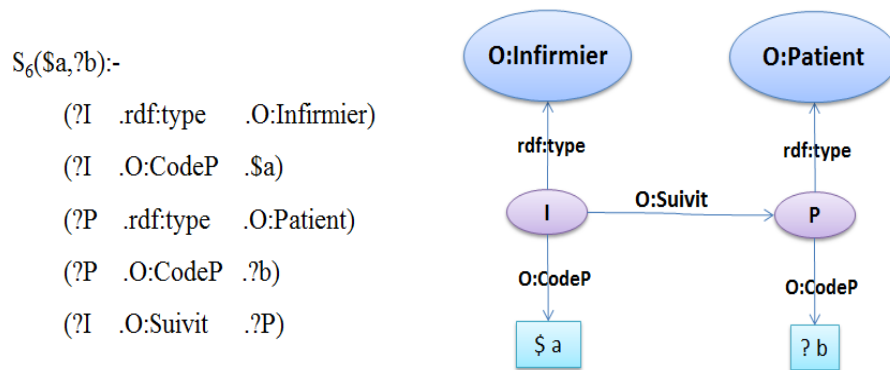


FIGURE 3.10 – Graphe RDF du service S_6

Le service S_7 donne les rapport (b) d'un patient (a).

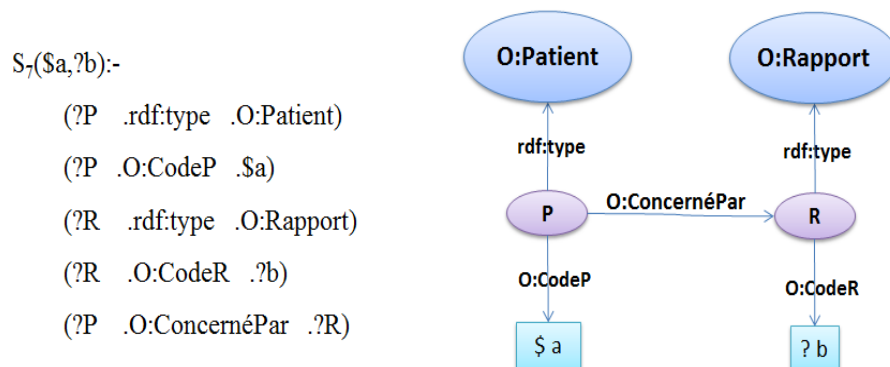


FIGURE 3.11 – Graphe RDF du service S_7

Le service S_8 donne les médecins (b) traitant un patient (a).

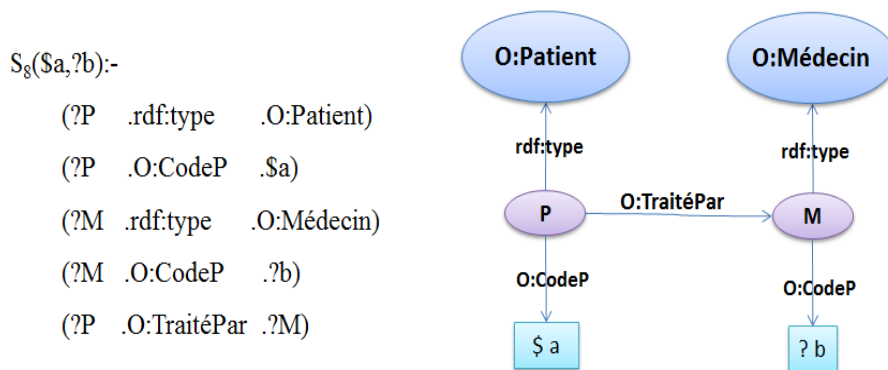


FIGURE 3.12 – Graphe RDF du service S_8

Le service S_9 donne les médecins (b) travaillant dans un service (a).

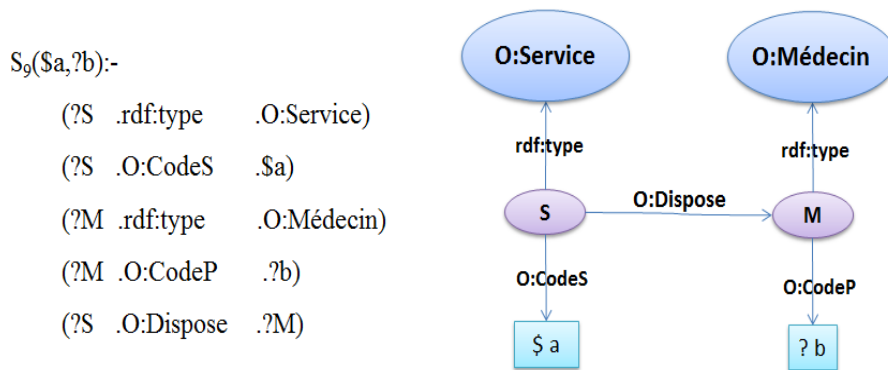


FIGURE 3.13 – Graphe RDF du service S_9

Le service S_{10} donne les rapports (b) liés à un rapport (a).

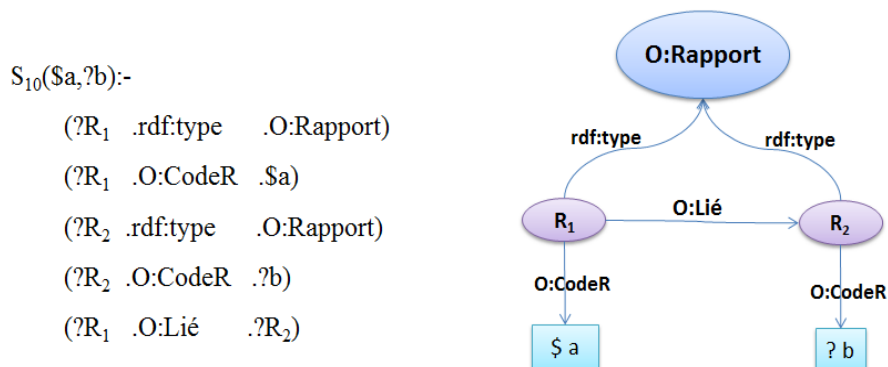


FIGURE 3.14 – Graphe RDF du service S_{10}

Le service S_{11} donne les pièces jointes images (b), vidéos (c), dicom (d) d'un rapport (a).

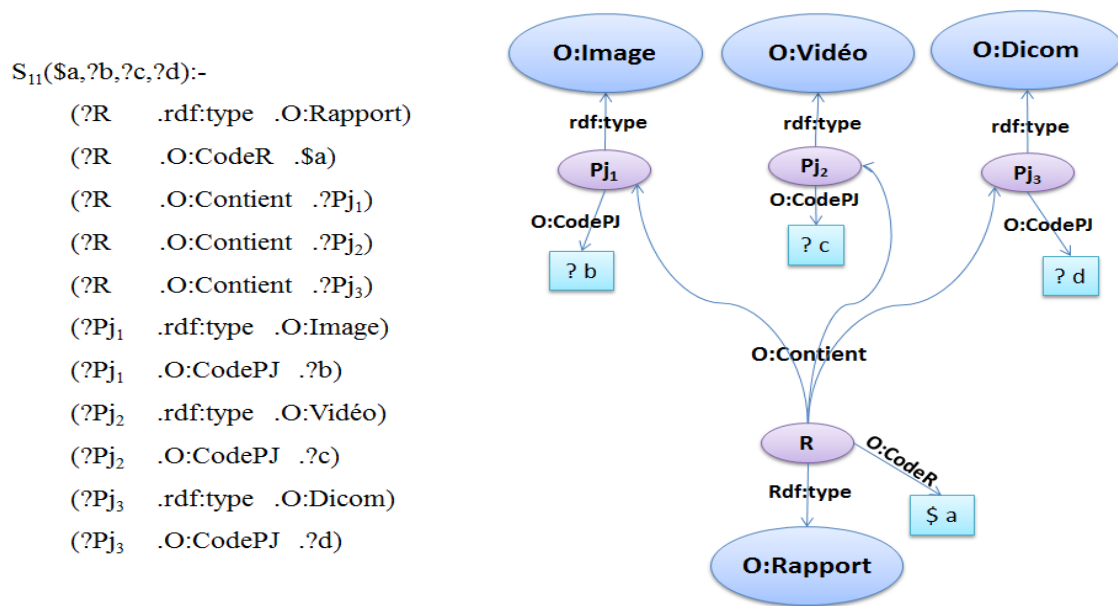


FIGURE 3.15 – Graphe RDF du service S_{11}

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:subClassOf 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.

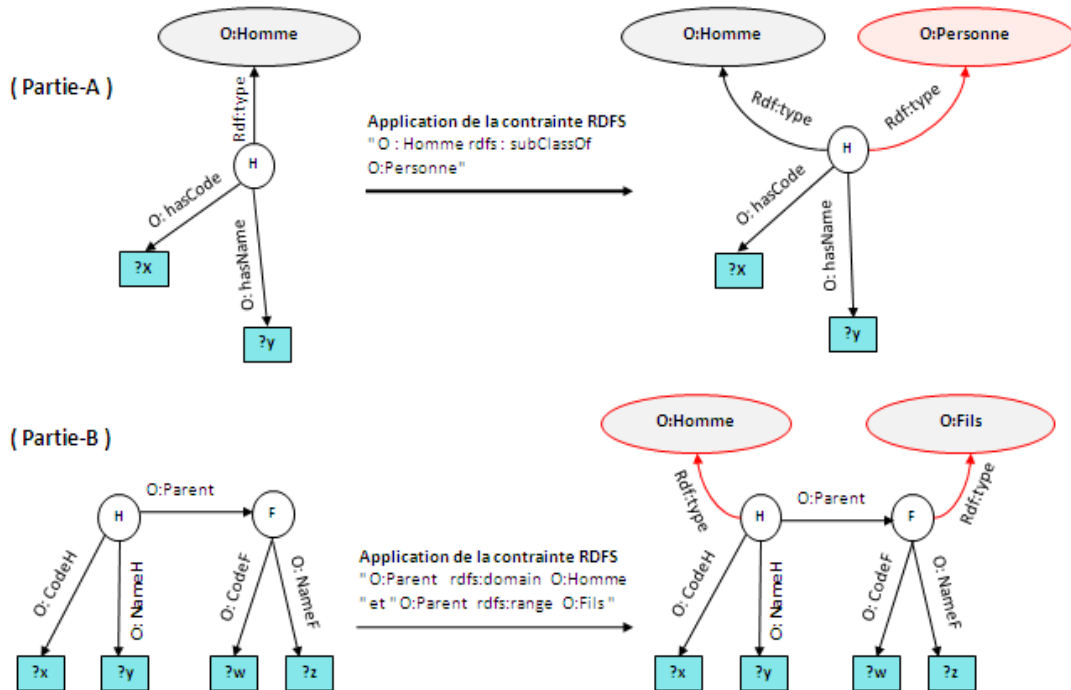


FIGURE 3.16 – Application des contraintes sémantiques RDFS

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

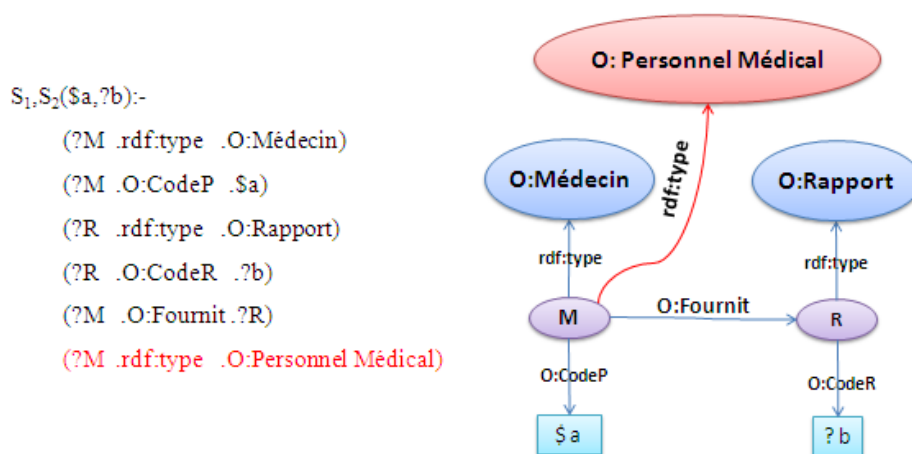


FIGURE 3.17 – Graphe RDFS des services S_1 et S_2

$S_3(\$a,?b):-$

(?P .rdf:type .O:Patient)
 (?P .O:CodeP .\\$a)
 (?P .O:Souffre .?Ma)
 (?Ma .rdf:type .O:Maladie)
 (?Ma .O:CodeM .?b)
 (?P .rdf:type .O:PersonnelMédical)

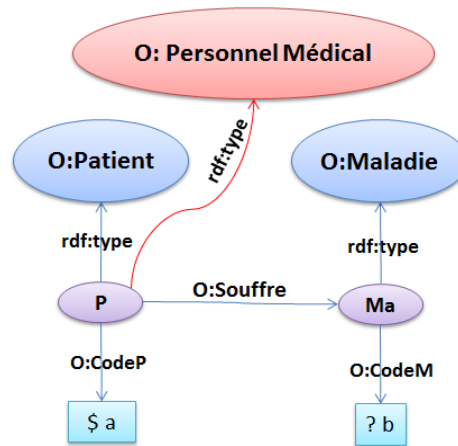


FIGURE 3.18 – Graphe RDFS du service S_3

$S_4,S_5(\$a,?b):-$

(?S .rdf:type .O:Service)
 (?S .O:CodeS .\\$a)
 (?I .rdf:type .O:Infirmier)
 (?I .O:CodeP .?b)
 (?S .O:Dispose .?I)
 (?I .rdf:type .O:PersonnelMédical)

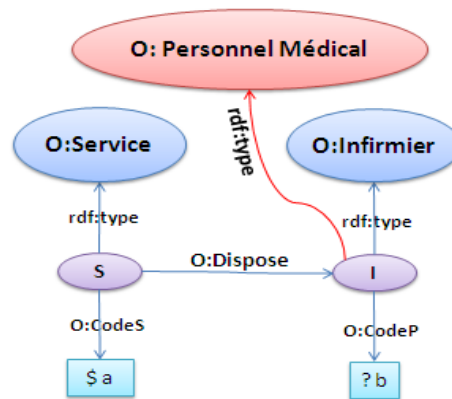


FIGURE 3.19 – Graphe RDFS des services S_4 et S_5

$S_6(\$a,?b):-$

(?I .rdf:type .O:Infirmier)
 (?I .O:CodeP .\\$a)
 (?P .rdf:type .O:Patient)
 (?P .O:CodeP .?b)
 (?I .O:Suivit .?P)
 (?I .rdf:type .O:PersonnelMédical)
 (?P .rdf:type .O:PersonnelMédical)

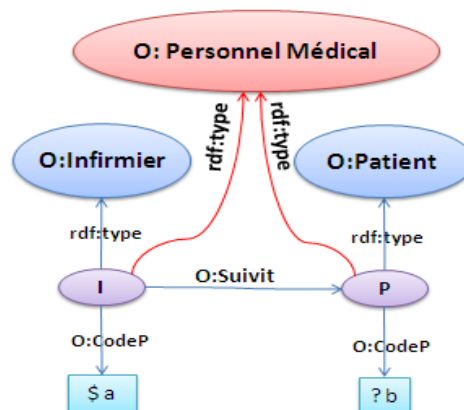


FIGURE 3.20 – Graphe RDFS du service S_6

$S_7(\$a,?b):-$
 (?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)

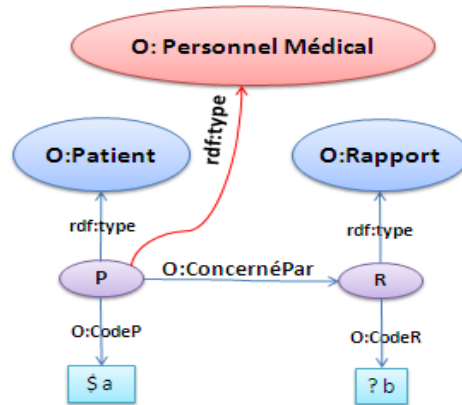


FIGURE 3.21 – Graphe RDFS du service S_7

$S_8(\$a,?b):-$
 (?P .rdf:type .O:Patient)
 (?P .O:CodeP .\\$a)
 (?M .rdf:type .O:Médecin)
 (?M .O:CodeP .?b)
 (?P .O:TreatéPar .?M)
 (?P .rdf:type .O:Personnel Médical)
 (?M .rdf:type .O:Personnel Médical)

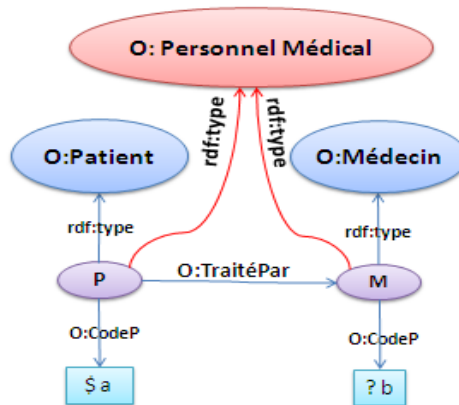


FIGURE 3.22 – Graphe RDFS des services S_8

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

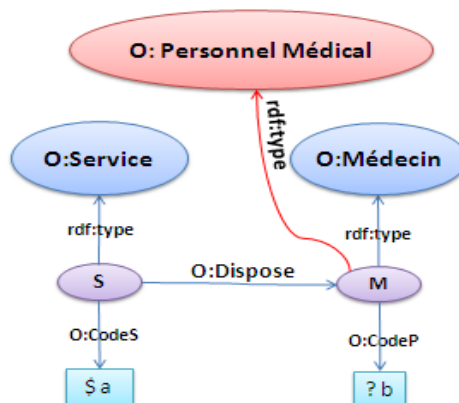
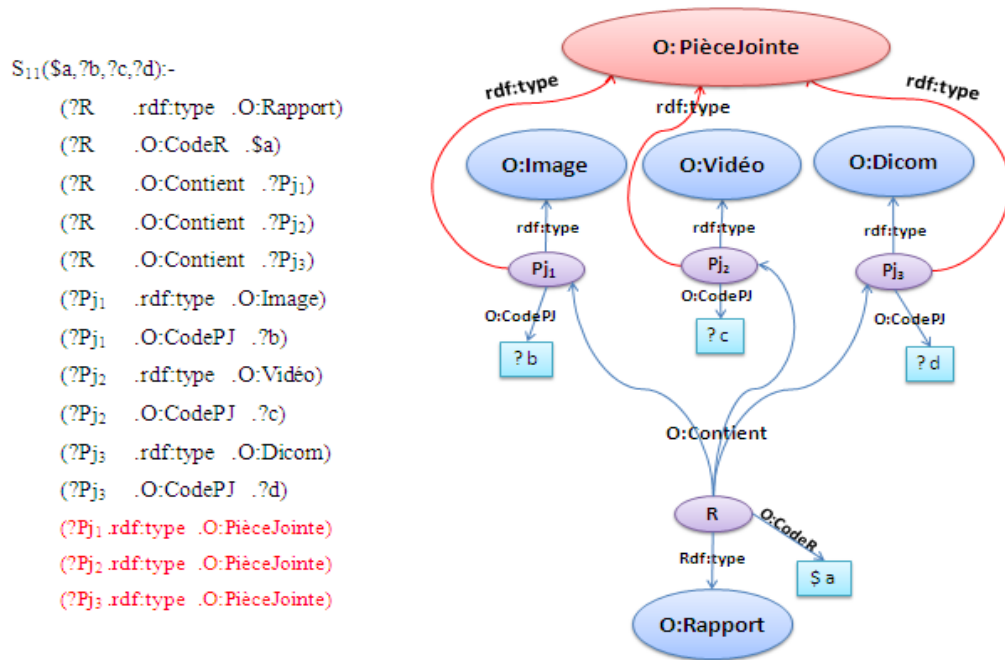


FIGURE 3.23 – Graphe RDFS des services S_9

FIGURE 3.24 – Graphe RDFS du service S_{11}

- 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 identifier 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 $F_1(\text{CodeM})$ 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 : $F_1(\text{CodeM}), F_2(\text{CodeP}), F_3(\text{CodeR}), F_4(\text{CodeS})$ et $F_5(\text{CodePJ})$.

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 $V = V_1, V_2, \dots, V_i$. 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** $\beta : G_Q \rightarrow G_V$ 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 projette 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 projette 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 incluse dans V_i tel que les Nœuds Classe liés par P dans V_i correspondent aux Nœuds 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 projette les DataType Properties utilisés dans la fonction skolem de chaque Nœud Classe lié par P ; ou
- (b) V_i couvre les Nœuds 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 projette 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 correspond 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 $S_2.M$.

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’ils 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.