Protection des données personnelles lors de la composition des services DaaS pour Mashup

Building Privacy-Aware DaaS Services Mashup

Salah-Eddine Tbahriti[1], Mahmoud Barhamgi[1], Nabila Benharkat[2], Chirine Ghedira[1], Djamal Benslimane[1], Michael Mrissa[1]

[1] LIRIS UMR5205, Université Claude Bernard Lyon1, 69622 Villeurbanne, France
prenom.nom@liris.cnrs.fr

[2] LIRIS UMR5205, INSA de Lyon, 7 av. Jean Capelle, 69621 Villeurbanne, France
nabila.benharkat@insa-lyon.fr

Résumé.

La technologie *Mashup* est l'une des applications émergentes du Web 2.0. Les applications mashups sont des applications hybrides dans lesquelles au moins deux services Web sont composés dans l'objectif de créer un nouveau service Web. Cette composition permet d'apporter une valeur ajoutée et présente ainsi une solution prometteuse pour l'intégration des données hétérogènes échangées par ces services. Cependant, les mashups rencontrent plusieurs limitations qui empêchent leur émergence. D'une part, les processus d'intégration actuels utilisés pour la création de *mashup* requièrent une implication considérable du concepteur humain, lequel est plus au moins familier avec les techniques de composition des services Web. D'autre part, la composition de plusieurs services diffusant des données peut révéler des informations personnelles confidentielles et enfreindre la vie privée des individus concernés par ces données. Dans cet article, nous présentons une approche de composition de services Web pour les concepteurs de Mashup de données où l'implication du concepteur se limite seulement à formuler les requêtes. Nous proposons aussi un modèle de confidentialité des données personnelles qui est exploité dans l'approche de composition.

Mots clés. Services DaaS, vues RDF, Composition de services, Confidentialité des données personnelles

1. Introduction

Le paradigme *Mashup* vise à utiliser et intégrer des sources de données et des fonctionnalités d'API (Interfaces de programmation) fournies par différentes applications en une seule application Web composite [1] [37] [39]. Pour cela, il agrège les informations de toutes les APIs qui le composent pour les publier a travers un nouveau service Web. En effet, la mise en place de nombreuses APIs dites ouvertes a permis le développement des mashups. Un des exemples le plus courant de mashup est *Google Maps* [38]. Le système *Trulia* aux Etats-Unis est un autre exemple de service mashup. Il intègre le service de Google Maps pour localiser un bien immobilier. Un autre exemple de mashup est le fameux système *ChicagoCrime* [42] qui recense le nombre de crimes commis dans la ville du Chicago aux États-Unis. Plus particulièrement, le *mashup de données* est un type de mashup qui permet de composer des informations de plusieurs sources de données fournies par le biais de services Web. Ce type de services est connu sous le nom de *DaaS* data-as-a-Services [1] [2].

Si les mashups de données gagnent du terrain aujourd'hui dans plusieurs domaines (e-business [4], e-science [3]), ils se limitent encore à une intégration d'informations non complexes (c'est-à-dire, représentées par types de des données simples). Nombreux sont les défis qui restent à résoudre aussi bien du coté des concepteurs (ou créateurs) de mashup que du coté des fournisseurs de données (c'est-à-dire, organisations qui développent les services

Web donnant accès aux données). Comme nous le verrons dans les sections 2.1 et 2.2 les solutions mises en place se limitent à effectuer manuellement un ensemble de tâches pour obtenir la composition des données souhaitée. Ces tâches comportent entre autres: la sélection des services DaaS pertinents pour l'application mashup à réaliser, la cartographie de leurs entrées et sorties (l'ajout probablement de certains services de médiation lorsque les entrées/sorties d'un service ne correspondent pas à celles des autres services) et le traitement des opérations intermédiaires (par exemple, les opérations de jointures, intersection) lors de la composition. Ces tâches peuvent s'avérer assez fastidieuses pour un concepteur non-expert. En outre, ces solutions sont généralement développées à l'aide de certains langages de programmation tels que JavaScript. Du côté service fournisseurs de données, la solution adoptée pour l'intégration des données soulève de nombreux problèmes notamment, ceux qui sont liés à la qualité et la confidentialité des données [5].

En effet, si la technologie des services DaaS a considérablement contribué à rendre l'information plus facilement accessible, répondant ainsi aux besoins des différentes communautés d'utilisateurs (statisticiens, médecins, chercheurs,…etc.), elle a cependant engendré en parallèle de nouveaux risques au regard de la confidentialité des données. Pour s'en persuader, il suffit de considérer la gigantesque base d'informations qu'il est possible de construire sur les individus en croisant les données historiques accumulées par l'ensemble des objets constituant une application mashup (état de santé, contenu des tiroirs, heures d'arrivée à la maison, déplacements, sites web visités, achats effectués par Internet, etc.). L'individu n'a plus conscience de la façon dont ses données personnelles sont collectées, traitées puis diffusées et même les informations les plus anodines peuvent être sujettes à interprétation (par exemple, une mauvaise alimentation est un facteur de risque pour une compagnie d'assurance). Ce constat est confirmé par une étude de *IBM-Harris* selon laquelle 94% des citoyens américains considèrent qu'ils ont perdu tout contrôle sur l'utilisation des informations les concernant [31]. En parallèle, l'étude menée dans [30] pour analyser les comportements des individus envers les services Web montre que seulement 27% des individus acceptent de divulguer leurs données personnelles à des services n'offrant aucune protection.

Le véritable verrou serait alors la standardisation de cette solution de mashup de données par les fournisseurs et les industriels avec la garantie que l'utilisation d'une telle technologie n'enfreigne pas la vie privée des individus concernés. Pour cela, il s'agit de tenter d'éliminer le risque (a défaut le minimiser au maximum) qu'un fournisseur n'ait plus le contrôle sur *l'usage de sa donnée* et/ou sur les actions qu'un tiers pourrait appliquer sur ses données.

1.1 Exemple de motivation

Considérons dans ce qui suit le scénario suivant pris dans le domaine médical. Considérons que le médecin *Alice* souhaite étudier les effets d'un médicament donné sur le taux de cholestérol des patients. *Alice* dispose des droits d'accès à un ensemble de services (cf. la table I). Ces services permettent d'accéder aux dossiers médicaux électroniques (DME) des patients. Ces services sont fournis par les différents établissements de santé dans le cadre d'un environnement collaboratif privé. Nous supposons aussi que les données de l'individu *Cathy* sont accessibles via les services de la table I. *Cathy* a été invitée (par chaque établissement de santé concerné) à exprimer ses contraintes sur l'utilisation de ses données en termes de confidentialité. Ainsi *Cathy* accepte de partager uniquement les résultats de ses tests médicaux et les données sur ses maladies (i.e., elle refuse de divulguer toute information liée à son *nom* et son *adresse*. *Mike*, un autre patient, accepte quant à lui de divulguer toutes ses informations personnelles et médicales avec des organismes scientifiques à des fins de recherche. Pour réaliser son étude, *Alice* utilise les services de la table I comme suit: elle invoque S_1 et S_2 avec un médicament donné, et avec les patients retournés elle invoque S_4

pour récupérer leurs informations personnelles. Puis elle invoque S_3 pour récupérer les tests dont la valeur = *cholestérol*.

1.2 Challenges

Dans cet exemple, *Alice* est confrontée à deux défis. Premièrement, elle doit être en mesure d'assimiler la description et la sémantique des services DaaS afin d'identifier correctement les services pertinents pour résoudre sa requête. En effet, différents services peuvent avoir la même description mais ils sont sémantiquement différents. Par exemple, les services S_5 et S_6 ont les mêmes paramètres en entrée «$*a*» et en sortie «?*b*» (c.-à-d., des patients et des maladies, respectivement), S_5 renvoie les maladies d'un patient alors que S_6 renvoie la ou les maladie(s) contre la(les)quelle(s) le patient est vacciné. Deuxièmement, *Alice* doit construire l'application mashup de données correspondante. Pour cela, elle doit sélectionner les services S_1, S_2, S_3 et S_4, déterminer leur ordre de composition et construire le plan d'exécution de mashup, ce qui est loin d'être une tâche prioritaire et encore moins facile pour un utilisateur non expert comme *Alice*.

Au-delà de ces problématiques, la conception d'une telle application devrait être soumise à des contraintes de confidentialité des données personnelles imposées par les fournisseurs de données. Ces contraintes traduisent les conditions d'usage des données (telles que l'objectif d'usage, les utilisateurs potentiels, le temps d'utilisation, etc.) et/ou les contraintes des patients en termes de confidentialité par rapport aux données les concernant. Les modèles de contrôle d'accès classiques [29] [28] ne peuvent pas conserver la confidentialité des données personnelles. En effet, si ces modèles sont plus au moins capables d'empêcher les utilisateurs non autorisés d'accéder à des données, ils ne peuvent contrôler les actions menées par les utilisateurs autorisés sur ces données. En d'autres termes, la *sécurité* ou le *contrôle d'accès aux données* traite les autorisations autour des données, tandis que *la protection de vie privée* s'intéresse aux problèmes de l'*usage* des données personnelles relatives aux individus. Par ailleurs, dans certains cadres d'application mashup, la divulgation des données personnelles devient nécessaire afin d'accomplir certaines tâches (le principe du *the need to know*). Par exemple, dans le cas des recherches épidémiologiques, toute approche de confidentialité devrait tenir compte des objectifs fonctionnels d'une application mashup. Par conséquent la protection des données personnelles est un problème plus complexe comparé à celui du contrôle d'accès. La complexité vient, d'une part du fait de l'équilibre qu'il faut trouver entre l'utilité de divulguer une donnée et le risque engendré par une *seconde utilisation malhonnête* de cette donnée et d'autre part de la complexité de l'approche en soi à mettre en place pour contrôler l'usage de ces données. Cela est d'autant plus complexe lorsque les sources de données sont de nature hétérogènes et encapsulées dans des services Web.

1.3 Contributions

Dans cet article nous proposons une approche d'intégration de données basée sur la composition des services Web DaaS. Dans un second temps nous proposons une approche déclarative pour la protection de données personnelles lors de la composition des services. Nous résumons ci-dessous nos principales contributions:

- Modélisation sémantique des services DaaS à base des vues RDF. Une vue RDF permet de représenter la sémantique d'un service en utilisant des concepts et des relations dont la sémantique est formellement définie dans des ontologies du domaine. En effet, l'une des caractéristiques des vues RDF est la possibilité de relier les entrées/sorties d'un service DaaS par une relation sémantique,
- Proposition d'un mécanisme de composition de services DaaS pour répondre aux requêtes, basé sur la réutilisation d'une technique de réécriture de requête,

- Spécification d'un modèle de confidentialité permettant d'appliquer les contraintes de confidentialité sur les données personnelles lors de la composition des services.
- Développement d'un outil de composition incorporant le modèle de confidentialité dans le cadre d'une application médicale.

Le reste de cet article est organisé comme suit. Dans la section II, nous discutons les principales approches mashup ainsi que les modèles de protection des données personnelles. Dans la section III, nous présentons une approche sémantique pour la modélisation des services DaaS et un modèle de politique de confidentialité des données personnelles. Dans la section IV nous décrivons notre approche de composition. Il s'en suit une évaluation de cette approche, dans le cadre d'une application médicale qui est proposée en section V. La conclusion et quelques travaux futurs sont présentés en section VI.

Service	***Sémantique opérationnelle du service***
S_1 ($a, ?b)	Retourne les patients identifiés par «*b*» qui ont pris le médicament identifié par «*a*»
S_2 ($a, ?b)	
S_3 ($a, ?b, ?c)	Retourne les données personnelles (nom «*b*» et adresse «*c*») d'un patient identifié par «*a»*
S_4 ($a, ?b, ?c)	
S_5 ($a, ?b)	Retourne les noms de maladie «*b*» d'un patient «*a*»
S_6 ($a, ?b)	Retourne les noms de maladies «*b*» contre lesquelles un patient «*a*» est vacciné

Table. 1 Description des Service Web DaaS

2. État de l'art

2.1 Les mashups

Plusieurs approches mashup ont été proposées dans le domaine académique et industriel [9], [10], [11] avec l'objectif de faciliter la réutilisation et la composition des services Web pour créer une valeur ajoutée. Des exemples de mashup sont *Yahoo Pipes* [6], *Google Mashup Editor* [7], *Intel Mash Maker* [8]. Ils permettent aux concepteurs lambda de créer facilement des applications mashup sans aucune programmation en dur. En effet, les concepteurs peuvent simplement faire glisser et déposer les services, les opérateurs, qu'ils veulent avoir dans l'application mashup. Bien que les avantages des mashups tels que souplesse et simplicité d'utilisation, rapidité de mise en œuvre soient évidents, les inconvénients sont tout aussi évidents. En effet, à partir du moment où l'on intègre des modules «ficelés» de fournisseurs externes de services, on ne maîtrise ni la qualité ni la pérennité. Cela comprend l'implication de l'humain dans le choix des services adéquats, la cartographie de leurs entrées et sorties et sans doute l'ajout de certains services de médiation. Ces approches s'intéressent à l'intégration fonctionnelle et évitent le problème fondamental de l'automatisation des différentes étapes liées à la fourniture d'un service Web (par exemples, découverte, sélection et composition) puisqu'elles limitent l'usage des services aux utilisateurs humains plutôt qu'aux machines. En effet, de nombreuses connaissances, indispensables pour l'automatisation de la composition de services, sont absentes ou décrites selon une syntaxe interprétée et exploitée uniquement par des humains. Il en résulte un rôle prédominant pour le

programmeur humain. Il semble donc nécessaire de proposer une approche de composition automatique des services qui rejoint les préoccupations à l'origine du Web sémantique, à savoir comment décrire formellement les connaissances de manière à les rendre exploitables par des machines. La sélection et la composition des services doivent être réalisées de façon transparente et automatique. L'objectif est alors de permettre à l'utilisateur d'exprimer, de manière *déclarative*, seulement les informations qu'il souhaite avoir comme résultat de l'exécution de l'application mashup, ensuite la composition des services se fera automatiquement et de manière complètement transparente.

2.2 Composition des services Web

L'objectif de la composition de services est de créer de nouvelles fonctionnalités en combinant des fonctionnalités offertes par d'autres services existants, en vue d'apporter une valeur ajoutée. Étant donnée une spécification de haut niveau des objectifs de l'application mashup à réaliser, la composition de services implique la capacité de sélectionner, composer et de faire collaborer des services existants. Parmi les travaux proposés, beaucoup adoptent la notion de processus métier «traditionnels» qui compose des services de type SaaS (*Software-as-a-Service Web services*). En effet, les travaux dans [12] [13] [14] définissent la composition comme un processus exécutable permettant de réaliser un ensemble d'actions. Un moteur d'exécution, un service Web jouant le rôle de chef d'orchestre, gère l'enchaînement des services Web par une logique de contrôle. Pour concevoir une orchestration de services Web, il faut décrire les interactions entre le moteur d'exécution et les services Web. Ces interactions correspondent aux appels, effectués par le moteur, et aux actions proposées par les *services Web composants*. Les algorithmes de composition, utilisés dans ces approches (largement inspirés des techniques de planification en IA) considèrent les services comme des actions et supposent ainsi que les fonctionnalités d'un service Web peuvent être modélisées selon le modèle *IOPEs* [15]. Cette hypothèse rend ces approches inapplicable pour les services DaaS car, d'une part le modèle *IOPEs* ne permet pas une représentation fidèle des relations sémantiques qui peuvent exister entre les entrées et les sorties d'un service DaaS et d'autre part les services DaaS ne peuvent être considérés comme des actions mais plutôt des interfaces offrant uniquement l'accès à des sources de données et n'exécutent aucune fonction qui change l'état du système. Le modèle à base des vues RDF nous semble le plus adapté pour représenter les relations sémantiques entre les entrées et sorties d'un service DaaS. L'approche décrite dans [34] offre un cadre formel pour formuler des requêtes en langage SQL. Chaque service est modélisé comme une relation $WS(a_1,.., a_n)$, où les a_i ($1 \leq i \leq n$) sont les attributs de la relation qui représentent les entrées et sorties du service. Les concepteurs peuvent ainsi exprimer leurs requêtes directement en termes de relations. Cela permettrait de pallier une partie du problème d'optimisation pour la génération du mashup. Dans le même esprit, l'approche décrite dans [35] et [36] offre aux concepteurs de mashup la possibilité d'importer et de stocker les fichiers de description des services (par exemple, les fichiers WSDL) et de définir des vues sur ces fichiers. Les vues peuvent être définies directement sur les opérations des services ou au-dessus sur d'autres vues, auquel cas elles sont appelées des vues multi-niveau. Les concepteurs peuvent alors interroger les données accessibles par les services importés par la formulation de leurs requêtes (le langage SQL) sur les vues définies. Les concepteurs peuvent également améliorer les vues définies avec des contraintes de clé primaire qui sont exploitées pour optimiser le plan d'exécution du mashup. Néanmoins, les concepteurs de mashup sont supposés avoir assimilé la sémantique de la description de chaque service. En outre, ils sont censés capables d'importer les services adéquats pour construire le mashup; puis définir des vues sur ces services ainsi que les index. En réalité, ces tâches peuvent être réalisées par des concepteurs ayant un certain niveau d'expertise par rapport aux techniques de description de services. Par ailleurs, la modélisation

adoptée est sémantiquement pauvre car elle se limite uniquement à représenter les entrées/sorties d'un service.

2.3 Protection des données personnelles

Partout dans le monde, les gouvernements adoptent des lois spécifiques pour contrôler l'utilisation de données personnelles comme le '*Federal Privacy Act*' [26] aux Etats-Unis ou la directive pour la protection des données personnelles en Europe [27]. Le problème n'est cependant pas simplement législatif. Encore faut-il être capable de mettre au point les outils technologiques permettant de traduire et faire respecter les règles édictées. Plusieurs approches sont proposées visant à traduire ces lois en moyens technologiques convaincants garantissant leur application. Nous en citons les principales d'entre eux. L'approche proposée dans [20] est basée sur l'extension d'*OWL-S* pour, d'une part l'introduction des politiques de sécurité dans *OWL-S* et, d'autre part, l'introduction des ontologies afin d'annoter les paramètres d'entrée/sortie des services Web, relatifs aux contraintes de sécurité. Deux types de politique sont spécifiés ; une politique d'autorisation qui comprend les règles d'accès à une donnée, et une politique de contrôle des données personnelles pour spécifier sous quelle(s) condition(s) les données peuvent être échangées et aussi les usagers légitimes des ces données. Ces politiques sont intégrées dans les descriptions *OWL-S* (au niveau des modules *Profil* et *Process*). Dans l'approche [21], les principaux éléments visant à protéger les données personnelles lors des échanges des données via des services Web gouvernementaux, sont présentés. L'idée de cette approche est que toute transaction invoque essentiellement trois entités clés ; *l'individu*, *le service* et *une base de données*. Par conséquent, un modèle de protection de données privées doit comprendre trois dimensions : 1) *user-privacy* (l'utilisateur d'un service peut être un individu et/ou un service Web) : les individus utilisant un service pourraient exiger des niveaux différents quant à la préservation des leurs données personnelles en fonction de leur *perception de la sensibilité de la donnée* fournie à un service. 2) *service-privacy* : un service Web doit posséder sa propre politique de confidentialité. 3) *data-privacy* : différents services Web peuvent solliciter différentes données d'une même source. Ici les services demandeurs sont les concepteurs. Le service Web fournisseur doit être capable de fournir plusieurs vues aux différents demandeurs selon leur *data-privacy-profile*. D'autres approches comme le standard *P3P* [19] offre un moyen technique permettant aux individus d'être informés des politiques de confidentialité avant de confier des renseignements personnels. Il n'offre cependant aucun mécanisme qui permet de garantir le comportement des sites conformément à leurs politiques. Le modèle *XACML* [22] est un formalisme qui permet de fournir un moyen de standardiser les décisions de contrôle d'accès (aux ressources en format XML). Il est largement utilisé dans le contexte industriel pour exprimer des politiques d'autorisation d'accès. Cela peut s'expliquer par le fait que ce langage permet une interopérabilité syntaxique entre les différents systèmes d'authentification et d'autorisation. En revanche, il ne permet pas de résoudre des problèmes sémantiques de coopération entre les différents systèmes. Cette approche peut s'avérer peu performante dans les cas où le processus de récupération de la valeur d'un attribut est long. Par ailleurs, il serait important de réfléchir sur comment introduire des nouvelles règles permettant de spécifier l'usage des données demandées. Le modèle *EPAL* [23] permet de définir des politiques qui catégorisent ; *les données détenues par l'entreprise, les utilisateurs* et *les finalités d'utilisation de données*. Une politique est formulée par des règles sur chaque catégorie. Comme une règle *XACML*, une règle *EPAL* comporte un sujet, une action et une ressource avec une indication de permission ou d'interdiction, et elle comporte en plus une mention de finalité (c'est-à-dire l'objectif d'usage). Une règle peut aussi contenir des conditions et des obligations. D'autres approches de protection des données personnelles sont proposées dans le cadre des bases de données centralisées et fouille de données. Elles s'appuient sur les

techniques d'anonymisation [24] (par exemple, *k-anonymat*, *l-diversity, t-closeness,* etc.) ou encore les méthodes de cryptage [25]. Plusieurs problèmes demeurent encore ouverts et nécessitent davantage d'efforts d'investigation, notamment la prise en compte et la modélisation des connaissances de l'adversaire. En effet, une des vulnérabilités importantes du k-anonymat, par exemple, est sa faiblesse contre les attaques ciblant à utiliser des connaissances externes sur les attributs quasi-identifiants[1].

Dans la section suivante, nous commençons par spécifier comment modéliser les services DaaS de façon à ce cela permette par la suite de les composer de façon automatique tout en tenant compte des contraintes de confidentialité des données personnelles. Ce dernier point représente un vrai défi puisque les approches de composition précédentes accordaient très peu d'attention à ce problème de confidentialité.

3. Une approche déclarative pour la construction des mashups

L'objectif de notre travail est de proposer un système mashup permettant aux concepteurs non-experts de composer automatiquement des services sans se préoccuper des étapes de sélection et d'invocation (l'utilisateur spécifié seulement sa requête de façon déclarative). Figure-1 présente un aperçu global de l'approche mashup proposée. La première étape vers cette automatisation consiste à représenter sémantiquement les capacités des services DaaS (qui constituent une couche virtuelle au-dessus de sources de données hétérogènes). Pour cela, les services DaaS sont représentés sémantiquement grâce à une modélisation en *vues RDF* basée sur des ontologies de domaine. Les vues RDF sont ensuite utilisées pour annoter les fichiers de description de service (par exemple, les fichiers WSDL, SA-Rest, etc.). Notre approche exploite les techniques de réécriture de [32]. La requête initiale est modifiée pour intégrer les contraintes de confidentialité sur les données personnelles. Ces dernières sont spécifiées selon une politique de confidentialité du mashup. La requête est ensuite réécrite en termes de services disponibles en utilisant un algorithme de réécriture de requêtes. Cet algorithme exploite les annotations sémantiques que nous avons ajoutées dans les fichiers de description de service pour sélectionner les services qui correspondent à la requête. Le serveur de mashup se chargera de créer un plan d'exécution des services sélectionnés. Dans cette section, nous détaillons toutes ces étapes.

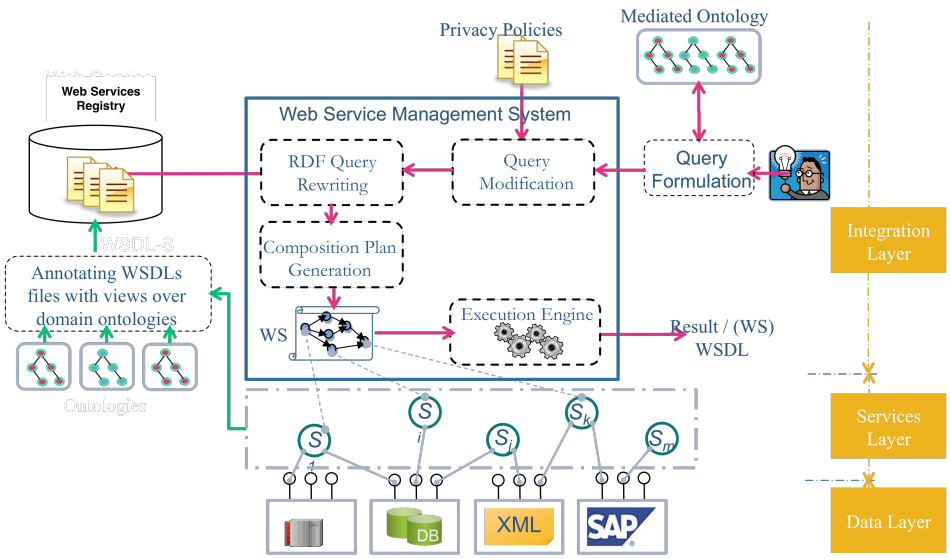


Figure 1. Vue globale de l'approche mashup

[1] Un ensemble d'attributs –non clés- permettant d'identifier indirectement un attribut clé d'une relation

3.1 Les requêtes sur mashup

Dans notre approche, les concepteurs de mashup n'ont pas besoin de sélectionner et composer les services manuellement. Ils spécifient seulement leurs besoins de données sous forme de requêtes conjonctives en langage SPARQL en termes d'une ontologie donnée appelée l'ontologie cible *Ω*. Formellement, les requêtes conjonctives s'écrivent comme suit :

Définition 1. Une requête mashup est représentée par : $Q(\overline{X}) :- < G(\overline{X},\overline{Y}), C >$ où $Q(\overline{X})$ est appelée l'entête de la requête (qui a la forme d'un prédicat relationnel), $\overline{X}$ est un tuple de variables distinguées et $\overline{Y}$ est un tuple de variables non-distinguées qui sont quantifiées existentiellement, $G(\overline{X}, \overline{Y})$ est le corps de la requête et C est un ensemble de contraintes appliqué sur la corps des variables exprimées sous la forme $x\ f\ a$ où x est une variable, $f \in \{=, >, <, \leq, \geq\}$ et a est une constante. ♦

La partie A de la figure 2 illustre une représentation graphique de la requête initiale de la section 1.1. Ainsi une requête est représentée par deux types de nœuds: les nœuds-classe et les nœuds-latéraux. Les nœuds-classe se réfèrent aux classes de l'ontologie *Ω* (par exemple, *M*, *P*, *T* sont des nœuds classe) et sont reliés par des propriété d'objet et représentent les variables existentielles dans la requête. Les nœuds-latéraux représentent les types de données (par exemple, *x, w, z*) et sont liés avec les nœuds-classe via les propriétés de type de données. Les nœuds-latéraux peuvent correspondre aux variables existentielles et distinguées dans une requête. Les concepts sont représentés par des ovales (par exemple, *Médicaments, Patients*). Les variables précédées par le symbole $, respectivement ?, représentent les entrées et les sorties d'un service DaaS.

3.2 Les services de données DaaS

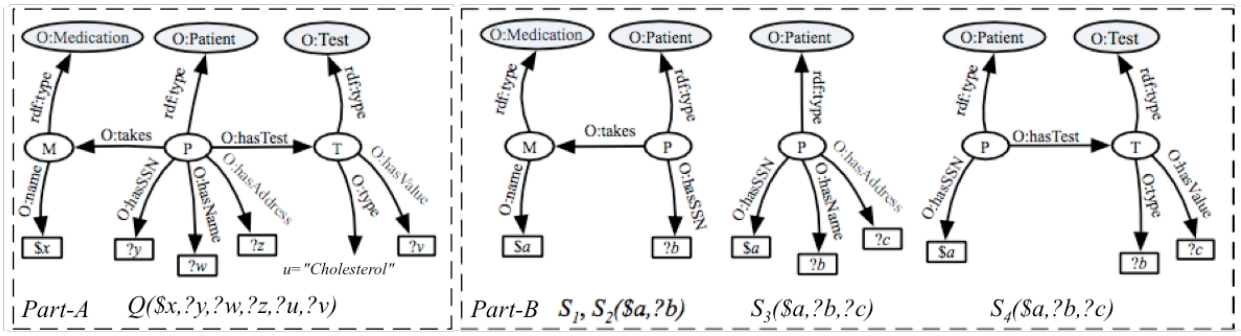
Contrairement aux services *SaaS*, la sémantique d'un service *DaaS* ne peut être fidèlement représentée à travers le modèle *IOPEs* (c'est-à-dire, les *entrées*, *sorties*, *conditions préalables* et *effets*). En effet, ce modèle ne permet pas de représenter la sémantique de la relation qui peut exister entre les entrées et sorties d'un service. C'est pour cette raison, que nous modélisons un service DaaS par une vue RDF paramétrée (notée *RPV*) en termes des concepts et relations de l'ontologie *Ω*. Formellement, un service DaaS est décrit comme suit :

Définition 2. Un service *DaaS* S_i est décrit sur *Ω* par un prédicat $S_i(\$\overline{X_i}, ?\overline{Y_i}) :- < RPV_i(\overline{X_i}, \overline{Y_i}, \overline{Z_i}), C_i >$ où :

- $\overline{X_i}$ et $\overline{Y_i}$ sont respectivement l'ensemble des variables d'entrée et de sortie (préfixés respectivement par $ et ?) de S_i. Ils sont appelés aussi l'ensemble des variables distinguées .
- $RPV_i(\overline{X_i}, \overline{Y_i}, \overline{Z_i})$ représente la relation sémantique entre les deux ensembles de variables d'entrée et de sorties. $\overline{Z_i}$ représente l'ensemble des variables existentielles reliant $\overline{X_i}$ et $\overline{Y_i}$. $RPV_i(\overline{X_i}, \overline{Y_i}, \overline{Z_i})$ est de la forme d'un ensemble de triplets RDF. Chaque triplet est une association (*sujet, prédicat, objet*),
- C_i est un ensemble de contraintes de valeurs sur les ensembles $\overline{X_i}, \overline{Y_i}$ et $\overline{Z_i}$. ♦

La partie B de la figure 2 illustre les graphes correspondant aux *RPVs* des services de la table 1. Chaque *RPV* nécessite une variable en entrée et retourne une variable en sortie. Ainsi, le

service S_1 ne peut pas retourner les *SSN* des patients qui ont pris un médicament sans préciser d'abord le *nom du médicament* en question.



Partie A **Partie B**

Figure 2. Représentation des services et requête par des vues RDF

3.3 Politique de confidentialité des données personnelles

Toute démarche de sécurité rigoureuse doit être inscrite dans une politique claire et documentée. Sa conception est donc une étape primordiale, qui consiste à identifier les objectifs de sécurité et à élaborer un ensemble de règles en fonction d'une analyse des risques. Ceci permettrait de minimiser le risque de dommages indésirables, le cas échant de pallier leurs effets et conduirait à garantir la protection de vie privée. Par conséquent, les fournisseurs de services *DaaS* devraient être en mesure de fournir une telle politique de confidentialité pour la protection des données personnelles spécifiant les conditions d'usage des services DaaS par rapport à ces données. Une politique de confidentialité des données personnelles (en anglais *privacy policy*) est définie sur un domaine ontologique par un ensemble de règles spécifiant l'usage des données personnelles, comme *à qui* une donnée personnelle peut être divulguée (*recipient*) et dans quel *objectif d'utilisation* (*purpose*). Une telle politique permet aussi de spécifier les conditions de divulgation comme le *consentement* de l'individu concerné sur la divulgation de sa donnée personnelle ou le principe du *besoin de savoir* (the *need to know*). Formellement, nous définissons une politique de confidentialité est définie comme suit :

Définition 3. Une politique de confidentialité est un quadruplet <*R, P, S, PC*> décrit sur une ontologie de domaine où :

- *R* est la classe des bénéficiaires (c'est-à-dire, les utilisateurs autorisés),
- *P* est l'objectif d'usage pour lequel une donnée personnelle sera utilisée,
- *S* est la classe des données personnelles,
- *PC* est un ensemble de couple (P_i, C_i), propriété-condition, qui signifie que la propriété P_i (valeur d'un objet) du concept *S* ne peut être divulguée que si la condition C_i est satisfaite. Chaque condition C_i peut être exprimée par rapport aux concepts et relations de l'ontologie selon le langage SPARQL. ♦

Par exemple, la règle *R1* (cf. figure 3-partie A)) spécifie que le *nom* peut être divulgué à un *chercheur* dans le cadre d'un *objectif de recherche* et à condition que l'individu concerné (c'est-à-dire le patient) ait donné son consentement pour cette divulgation (le même raisonnement est appliqué à l'*adresse*). *R1* est définie en utilisant les concepts *Patient* et *PatientPrivacyPreferences* de l'ontologie *Ω* (le concept *PatientPrivacyPreferences* permet de

modéliser les préférences d'un individu par rapport à la divulgation de ses données personnelles). La règle *R2* (cf. figure 3-partie B)) indique que la propriété *hasValue* du concept *Test* peut être divulguée à un *chercheur* dans le cadre d'un *objectif de recherche* et à condition que l'individu concerné ait donné son consentement pour cette divulgation. Les préférences de l'individu par rapport à la divulgation de ses données sur les tests médicaux sont modélisées par le concept *TestPrivacyPreferences*, le prédicat *hasTest-PrivacyPreferences* permet de lier le concept *Patient* au concept *TestPrivacyPreferences* dans Ω.

Dans la section suivante, nous détaillons notre approche de composition des services DaaS.

```
Rule-1:

[R :Researcher,
 P :Research,
 S :Patient,
 PC:{<hasName, "?P  rdf:type O:Patient,
                ?P  hasPrivacyPreferences  ?PP,
                ?PP rdftype O:PatientPrivacyPrefernces,
                ?PP hasPrivacyPrefName  'yes'">,
     <hasAddress, "?P  rdf:type O:Patient,
                   ?P  hasPrivacyPreferences  ?PP,
                   ?PP rdftype O:PatientPrivacyPrefernces,
                   ?PP hasPrivacyPrefAddress  'yes'">,
     }
]
```

Partie A

```
Rule-2:

[R :Researcher,
 P :Research,
 S :Test,
 PC:{<hasValue, "?P  rdf:type O:Patient,
                 ?P  hasTestPrivacyPreferences  ?TP,
                 ?TP rdftype O:TestPrivacyPrefernces,
                 ?PP PrefValue  'yes'">,
     }
]
```

Partie B

Figure 3. Règles de politique de confidentialité des données personnelles

4. Approche de composition des données via DaaS

Nous proposons une approche de composition basée sur trois étapes: (i) la réécriture de la requête mashup afin d'intégrer les contraintes de confidentialité, (ii) la réécriture de la requête mashup en termes de services DaaS disponibles et (iii) la construction automatique d'un plan de composition de mashup.

4.1 Intégration de contraintes de confidentialité dans la requête

Les requêtes sont formulées en langage SPARQL. Les conditions associées à chaque propriété *datatype* dans les règles de confidentialité sont formulées en requêtes RDF (requêtes SPARQL). Cela permet de les intégrer aisément à la requête initiale. Les conditions de confidentialités doivent être renforcées à un niveau de granularité très fin, c.à.d, le masquage d'une propriété ne doit pas impacter la divulgation des autres propriétés demandées par la requête. Ce niveau de granularité peut être réalisé en utilisant les clauses OPTIONAL du SPARQL [33]. La sémantique de cette clause est traduite comme suit: dans une requête RDF conjonctive, toutes les variables de la requête doivent être liées à des valeurs dans le graphe RDF de données correspondant afin que la requête n'échoue pas. Si une variable est définie comme OPTIONAL et s'il n'existe pas de valeurs avec lesquelles cette variable peut être liée (c'est-à-dire, il n'existe pas de triplets RDF dans le graph de données correspondant) alors cette variable est non-liée et aura la valeur «Null» dans les tuples retournées. En d'autres termes la requête retourne des valeurs pour les autres propriétés indépendamment de la propriété manquante qui va avoir la valeur «Null». La protection des données personnelles est appliquée au niveau de la propriété *datatype* (de chaque concept de Ω) en associant dans la

clause OPTIONAL chaque propriété *datatype* concernée avec les conditions de divulgation. Si ces conditions sont satisfaites la valeur de la propriété est alors divulguée. Par ailleurs, si l'une des conditions n'est pas satisfaite la valeur de la propriété concernée ne sera pas divulguée. Prenons l'exemple de la requête *Q1* décrite dans la figure 4-partie A. La partie B de la figure 4 illustre la réécriture de *Q1* pour intégrer les règles de confidentialité précédentes *R1* et *R2* comme suit :

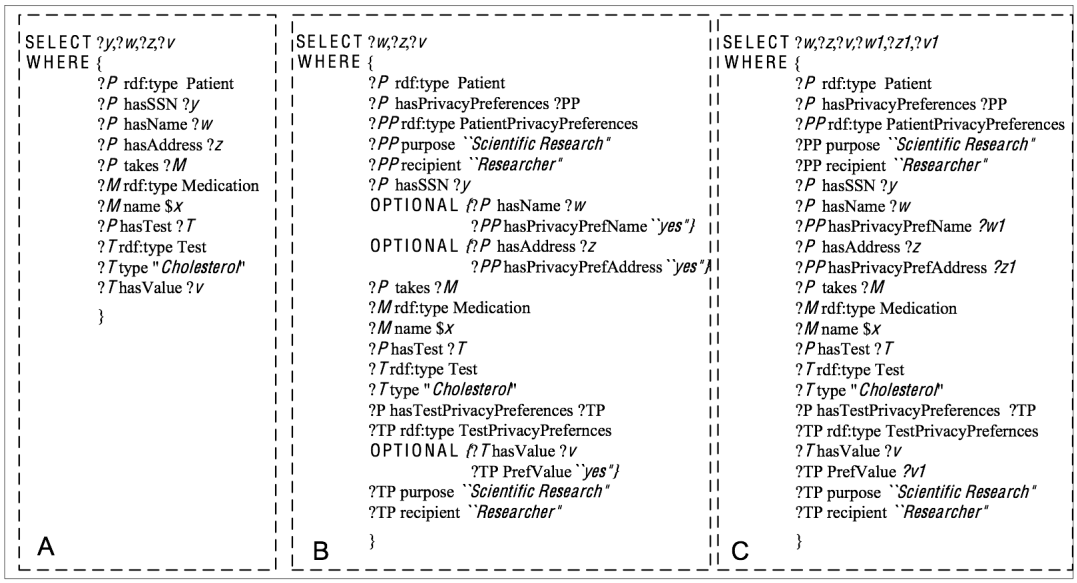


Figure 4. Les étapes de réécriture d'une requête

(i) la propriété «*hasSSN*» est une propriété *datatype*. Cette propriété ne doit pas être divulguée aux chercheurs, par conséquent la variable distingué *?y* est supprimée de la clause SELECT lors de la réécriture, (ii) la propriété «*hasName*» est une propriété *datatype* aussi et peut être divulguée pour les chercheurs si l'objectif d'utilisation est *recherche scientifique* (Scienfic Research comme décrit dans la requête). L'utilisateur de «*hasName*» doit avoir le consentement de l'individu concerné qui est décrit par «?PP *hasPrivacyPreName ''Yes''*» dans la clause OPTIONAL. Autrement dit, si la propriété «?PP *hasPrivacyPreName* » est affectée la valeur *''No''* la variable *?w* sera égale à «Null». Le même traitement est appliqué pour les propriétés «*hasAddress*» et «*hasValue*» (cette dernière représente la valeur du test médical). Dans les applications mashups, les requêtes ne sont pas évaluées directement sur les données du mashup mais plutôt sur les services qui encapsulent les sources de données. Ainsi, chaque requête est réécrite en termes de services DaaS disponibles. Dans la section suivante, nous présentons notre approche de réécriture à base de vues RDF. Nous considérons les requêtes conjonctives (c'est-à-dire, tous les triplets RDF dans la requête sont *implicitement* reliés par l'opérateur conjonctif ET).

La présence de la clause OPTIONAL rend la requête non-conjonctive (puisque des parties de la requête sont obligatoires et des autres parties sont facultatives). L'algorithme de réécriture que nous utilisons est destiné pour traiter seulement des requêtes conjonctives. Pour pallier à ça, nous transformons la requête que nous avons obtenue dans l'étape précédant (la requête dans la figure-4, B) en une requête conjonctive. Les conditions de la clause OPTIONAL seront alors mises dans la requête mais ne seront pas évaluées. Considérons par exemple la condition associée à la propriété *hasName* dans la première clause OPTIONAL. La condition exige que le patient soit d'accord pour divulguer sa donnée (c'est-à-dire donne « *hasPrivacyPrefName = Yes* »). Cette condition est alors insérée dans la requête mais sans ne

sera pas évaluée; Ainsi, nous ajoutons le triplet suivant à la requête ?PP *hasPrivacyPrefName* $?w_1$ et nous ajoutons la nouvelle variable $?w_1$ dans la clause SELECT. Par conséquent, la requête contient à la fois les variables demandées par la requête initiale (c'est-à-dire, les variables *w*, *z* et *v*) et la nouvelle variable w_1 (le même raisonnement est appliqué pour toutes les variables dans la clause OPTIONAL). A la fin de cette étape nous obtenons la requête illustrée dans la figure 4-partie C.

4.2 Réécriture de requêtes mashup

Dans [17] nous avons proposé un algorithme de réécriture à base de vues RDF. Étant donné une requête *Q* et un ensemble *V* de services DaaS (l'ensemble des services candidats) représentés par des vues RPVs $V = v_1, v_2, v_i$, cet algorithme permet de réécrire automatiquement *Q* en termes de V, chaque réécriture corresponde à une composition de services DaaS. L'objectif de cet algorithme est de calculer l'union des graphes RDF représentant ces services (notée par *GV*) qui couvre au maximum le graphe représentant *Q* (noté par *GQ*). Deux phases sont nécessaires pour cet algorithme:

- Phase-I: Mapping de *GQ* et *V*:
Dans la première phase, le graphe de la requête *GQ* est comparé à chaque RPV v_i dans *V* pour identifier les nœuds-classe et les propriétés de *GQ* couverts (partiellement) par v_i. Une table de mapping est construite au fur et à mesure contenant les différents graphes où chaque graphe couvre partiellement *GQ*.

Exemple 1. Considérons les services candidats suivants:

- Les services S_1 et S_2 : Le service S_1 couvre la propriété «*takes*». Les nœuds-classe $S_1.P$ et $S_1.M$ liés par cette propriété correspondent ainsi aux nœuds-class de Q_1 (de la figure 3.A). C'est-à-dire, $S_1.P$ correspond à $Q_1.P$ et $S_1.M$ correspond à $Q_1.M$). Les propriétés fonctionnelles de type de données des concepts «*Patient*» et «*Medication*» sont projetées par S_1 (c'est à dire elles correspondent à des variables distinguées de S_1). La propriété «*takes(*$Q_1.M$, $Q_1.P$*)*» est ainsi insérée dans la table de mapping (cf. table 2). Le même raisonnement est appliqué pour S_2.
- Le service S_3 : Ce service possède un nœud-classe $S_3.P$ qui peut correspondre à nœud-classe $Q_1.P$. Toutes les propriétés de type de données de $Q_1.P$ qui sont liées à des variables distinguées dans Q_1 sont également liées à des variables distinguées dans S_3. En outre, le service S_3 a la propriété *«hasSSN»* qui est liée à une variable distinguée dans sa vue RDF. Par conséquent, S_3 peut être utilisé pour couvrir $Q_1.P$.
- Le service S_4. La propriété «*hasTest*» du S_4 correspond à celle dans Q_1. Les nœuds-classe reliés par $S_4.P$ et $S_1.T$ correspondent aux nœuds-classe dans Q_1: $Q_1.P$ à $Q_1.T$. S_4 permet de lier les propriétés fonctionnelles des patients (c'est-à-dire, *«hasSSN»*) aux variables distinguées. Le service S_4 couvre la propriété «*hasTest*» et le classe nœud $Q_1.T$.

Considérons maintenant les services S_8 et S_9 (cf. figure 5). Les nœuds-classe du S_8 reliés par la propriété *«hasPatientPreferences»* correspondent aux nœuds-classe dans Q_1. S_8 ne reliant pas les propriétés fonctionnelles du concept «*PatientPrivacyPreference*» aux variables distinguées, il doit couvrir aussi le nœud-classe $Q_1.PP$, ce qui est possible. Le même raisonnement est appliqué pour le cas du S_9 en remplaçant le nœud-classe $Q_1.PP$ par $Q_1.TP$ et la propriété *«hasPatientPrivacyPreferences»* par *«hasTestPrivacyPreferences»*.

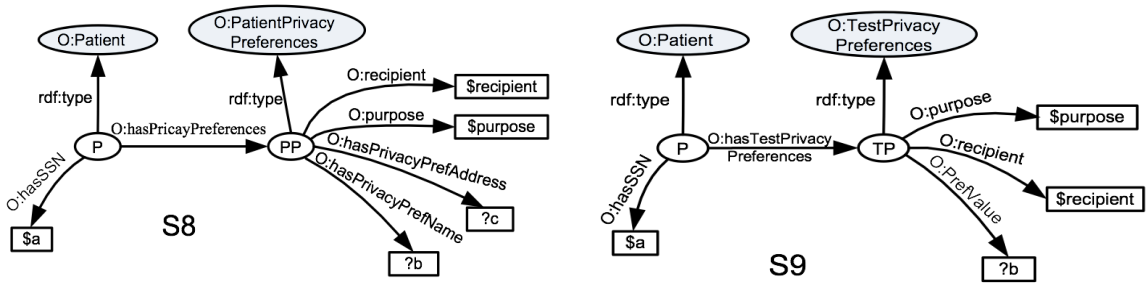


Figure 5. Services S_8 et S_9

Service	***Les nœuds-classe et propriétés couverts***
S_1 ($x, ?y)	*takes*(*P*, *M*)
S_2 ($x, ?y)	*takes*(*P*, *M*)
S_3 ($y, ?w, ?z)	*P*(*y*, *w*, *z*)
S_4 ($y, '*cholesterol*', ?v)	*hasTest*(*P, T*)*T*('*cholesterol*', *v*)
S_8 ($y, '*researcher*', '*research*', $?w_1$, $?z_1$)	*hasPatientPrivacyPreferences*(*P*, *PP*)*PP*(*'researcher'*, *'research'*, w_1, z_1)
S_9 ($y, '*researcher*', '*research*', $?v_1$)	*hasTestPrivacyPreferences*(*P*, *TP*)*TP*(*'researcher'*, *'research'*, v_1)

Table 2. Table de mapping

- Phase-II: Génération de la composition: Après la construction de la table de mapping, notre approche de réécriture explore les différentes compositions possibles des services dans cette table telle que une composition, notée par C = {S_1, S_2,…, Si}, est considérée comme une réécriture valide de *Q* si et seulement si: (1) C couvre l'ensemble des nœuds-classe et des propriétés dans Q, et (2) C est exécutable. Une composition est dite exécutable si tous les paramètres d'entrée nécessaires à l'invocation de ses services sont fournis ou peuvent être fournis par l'invocation de services primitifs dont leurs paramètres d'entrée sont fournis.

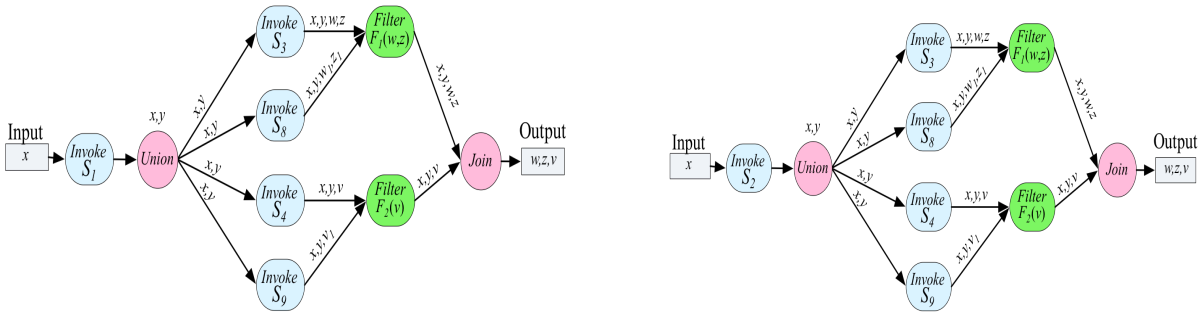
Exemple 2. Continuons sur notre précédent exemple 1, à partir de la table 2 deux compositions sont possibles C_1 = {S_1, S_3, S_4, S_8, S_9} et C_2 = {S_2, S_3, S_4, S_8, S_9}. Pour la composition C_1, uniquement le service S_1 ($x, ?y) est *invocable* au départ (ses paramètres sont liés). Une fois invoquée, la variable ?*y* devient alors disponible et les services S_3, S_4, S_8 et S_9 deviennent invocables. C_1 est alors exécutable est considérée comme une composition valide. Le même raisonnement est vérifié pour C_2 qui est donc valide.

4.3 Construction du mashup

A) L'ordonnancement des services dans le mashup

Les services d'une composition C valide sont reliés intuitivement selon un ordre particulier en fonction de leurs paramètres fonctionnels (c'est-à-dire, les paramètres d'entrées et sorties). Ainsi, si un service S_j a comme paramètre d'entrée ?*x* obtenu à partir d'un paramètre de sortie ?y d'un service S_i, alors le service S_j dépend fonctionnellement du service S_i. En d'autres termes, S_i est le prédécesseur de S_j dans C. Pour représenter l'ordonnancement des services dans C, nous définissons un graphe de dépendances, noté *GD*, (c'est-à-dire, un graphe orienté acyclique) dans lequel chaque nœud correspond à un service de la composition C et chaque arc entre deux nœuds correspond à une (ou plusieurs) paramètres de dépendance

entre les deux services. Ainsi le plan du mashup correspondra au graphe *GD*. La figure 6 présente le plan du mashup pour C_1 et C_2. Dans C_1,(cf. figure 6- partie A) S_1 fournit en sortie le paramètre *y* qui est le paramètre d'entrée pour les services S_3, S_8, S_4 et S_9, par conséquent, ces services doivent précéder S_1 dans le plan (pour C_2 -figure 6-partie B, le même ordonnancement est considéré avec comme différence S_2 à la place de S_1).



Partie A **Partie B**

Figure 6. Graphe de dépendance pour C_1 et C_2

B) L'application de la politique de confidentialité

Dans les étapes précédentes, les *propriétés datatype* qui sont nécessaires à l'évaluation des contraintes de la confidentialité ont été ajoutées à la clause SELECT; (la clause SELECT contient désormais les variables demandées par la requête initiale et les variables nécessaires à l'évaluation des contraintes de confidentialité placées aussi comme variables dans la requête initiale). Pour évaluer ces contraintes de confidentialité, nous augmentons le plan de mashup avec des operateurs de filtrage (appelés *Filter*) (ces opérateurs sont invokés lors de l'exécution du mashup). Pour chaque propriété sensible p_i (c'est-à-dire, une variable) contenu dans la requête initiale, les filtres utilisent les valeurs des propriétés ajoutés *P* pour évaluer les contraintes de confidentialité placées sur p_i. Des valeurs «Null » sont retournées quand la contrainte est évaluée à faux. Les operateurs de filtrages sont insérés à la sortie de chaque service retournant une donnée sensible. Nous définissons la sémantique de l'operateur *Filter* comme suit.

Définition 4. Soient T et T_p deux tables représentants respectivement les résultats de l'invocation d'un service avant et après l'application des contraintes de confidentialité. Soit t (respectivement t_p) un n-uplet dans la table de sortie T (respectivement T_p). Soit $t[i]$ (aussi $t_p[i]$) une colonne représentant une propriété associée à des contraintes de confidentialité, et *constraint*($t[i]$) une fonctionne booléenne permettant d'évaluer les contraintes associées à $t[i]$. Un n-uplet t_p est inséré à T_p comme suit :

```
Pour chaque n-uplet t ∈ T
      Pour i = 1 à n                /* n est le nombre de colonnes de T */
            Si const(t[i]) = true Alors t_p [i] = t[i]
                                  Sinon t_p [i] = null
Eliminer tous les n-uplets dans T_p qui contiennent que des valeurs nulles
dans leurs colonnes
```

Pour notre exemple de motivation, comme la figure 4 le montre, deux filtres F_1 et F_2 ont été insérés aux sorties des services S_3 et S_4 respectivement. Le filtre F_1 calcule les valeurs de *w* et *z* comme suit :

```
w = w Si w1 = 'yes', Sinon w = null
z = z Si z1 = 'yes', Sinon z = null
```

Le filtre F_2 calcule les valeurs de *v* comme suit :

```
v = v Si v1 = 'yes', Sinon v = null
```

Le plan de mashup obtenu après l'ajout des filtres de confidentialité représente le mashup qui sera retourné à l'utilisateur.

5. Prototype et évaluation

5.1 Mise en œuvre du système de composition

Nous avons mis en œuvre un système d'interrogation et de composition des services DaaS. L'architecture, illustrée en figure 7, est divisée en quatre couches. La première couche contient un ensemble de bases de données Oracle/MySQL qui stockent des données médicales. La deuxième couche est composée d'un ensemble de fonctions développées en Java. Chaque fonction a pour rôle d'exécuter des requêtes pour accéder à une base de données de la première couche et récupérer le résultat de ces requêtes. Ces fonctions sont exportées sous forme de services Web qui constituent la troisième couche de l'architecture. Nous avons utilisé le kit de déploiement fourni avec le serveur Web *GlassFish* pour créer et déployer ces services.

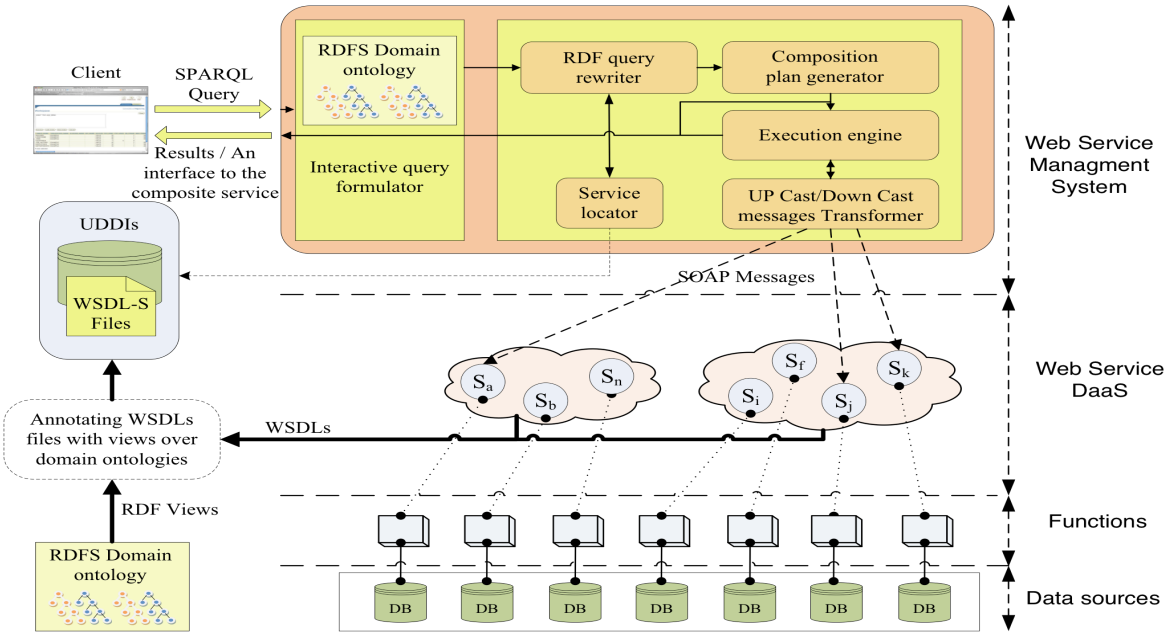


Figure 7. L'architecture du système de composition

Les fichiers de description WSDL-S des services Web situés en troisième couche sont annotés par les vues RDF. La sémantique de ces vues est interprétée selon les ontologies de domaine (décrites en RDFS). Les fichiers WSDL annotés sont publiés dans les registres de service Web. La couche supérieure est composée d'une interface graphique pour l'utilisateur du

système (appelé *GUI*) et d'un système gestion de Service (appelé *WSMS*). Cette interface est implantée en Java-Swing et permet aux concepteurs de formuler leurs requêtes qui sont exécutées par le système *WSMS*. Ce dernier est lui même composé de plusieurs modules :

- *Interactive Query Formulator*: permet d'aider les concepteurs à préciser leurs requêtes RDF (écrite en langage SPARQL) conformément à l'ontologie,
- *Service Locator*: permet de récupérer les fichiers des descriptions WSDL-S des services adéquats à partir des registres UDDI.
- *RDF Query Rewriter*: permet de mettre en œuvre l'algorithme de réécriture des requêtes RDF. Il détermine si les services proposés par le module *Service Locator* peuvent être utilisés pour répondre à la requête posée et retourne ainsi les compositions possibles.
- *Composition Plan Generator*: génère les plans d'exécution pour les compositions obtenues. Les plans générés sont envoyés pour exécution immédiate.
- *Execution Engine*: met en œuvre les différents opérateurs utilisés dans les plans d'exécution
- *UP-Cast/Down-Cast Messages Transformer*: transforme les messages échangés avec les services invoqués en cas de besoin.

La figure 8.1 montre l'interface utilisateur. Sur le côté gauche, le menu ontologie présente une vue arborescente de l'ontologie de domaine, le menu services présente les services stockés dans les registres de service.

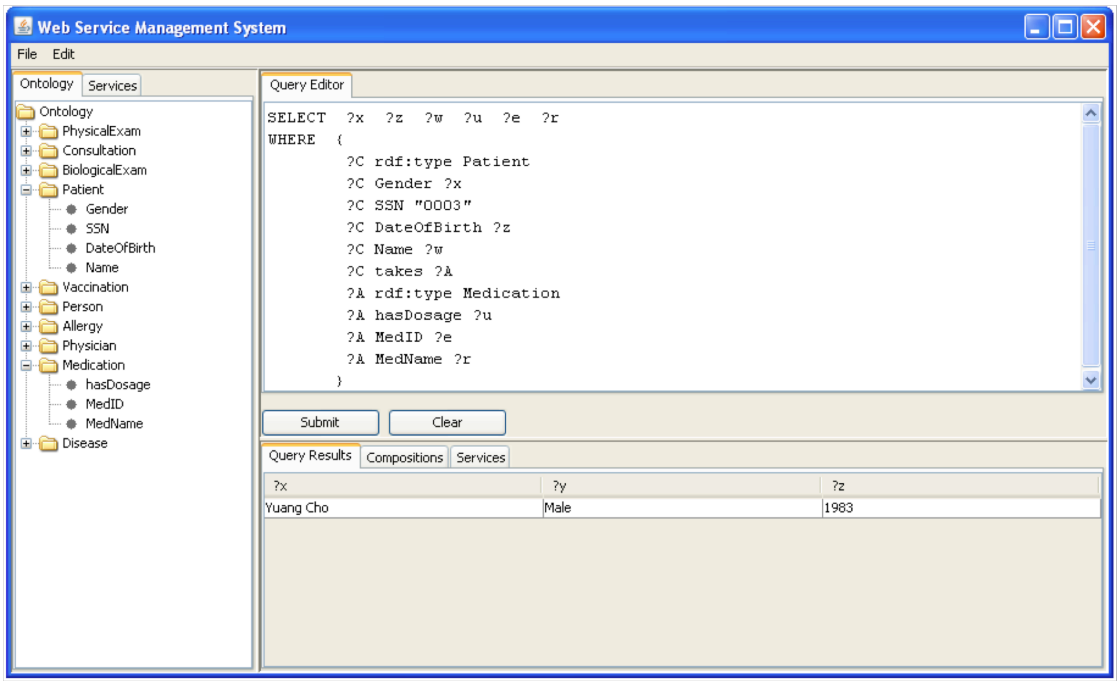


Figure 8.1. L'interface principale

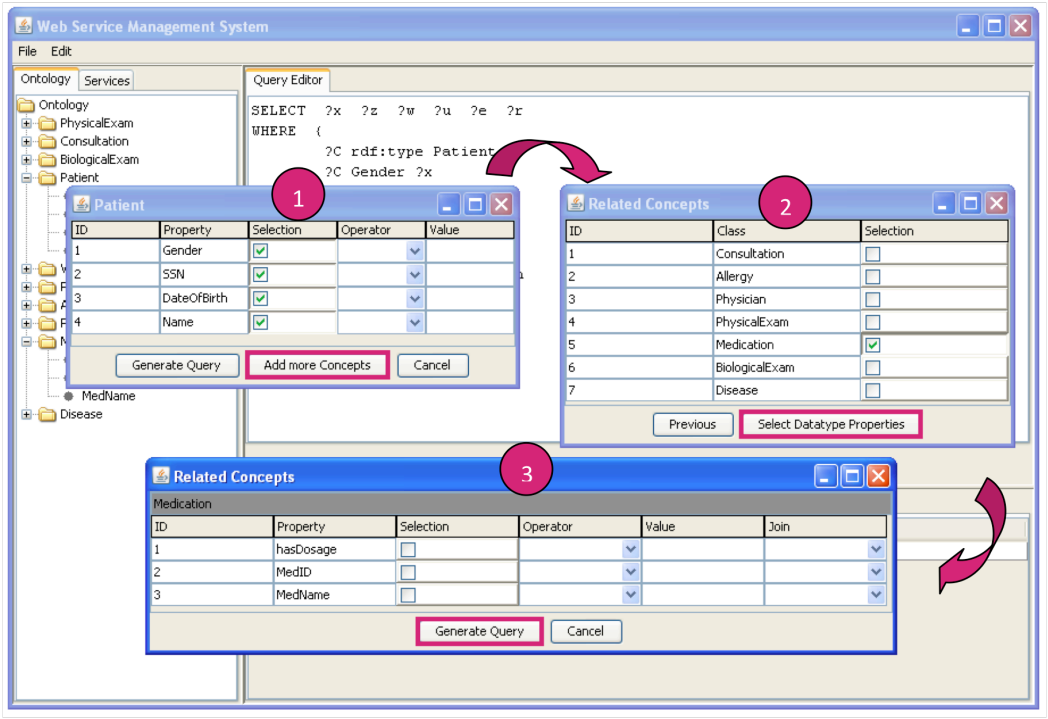


Figure 8.2. Les interactions lors de la formulation d'une requête

Lorsqu'un utilisateur veut formuler sa requête, il sélectionne les concepts qu'il souhaite ainsi que leurs propriétés en cliquant sur le bouton *Select Datatype properties* (cf. figure 8.2). Il peut alors cocher les propriétés souhaitées et la requête sera alors générée.

5.2 Expérimentations

Pour mesurer le paramètre passage à l'échelle de l'approche de composition et l'impact de l'application de politique de confidentialité sur le temps de réponse global, nous avons appliqué cette approche au domaine de la santé. Pour cela, un nombre de 411 services Web est généré sur 23 bases de données médicales différentes (bases de données Oracle) contenant des informations médicales (par exemple, des dossiers médicaux, allergies, etc.) relatives à plus de 30.000 patients. Ces services ont été déployés sur un serveur Web GlassFish. L'utilisation de ces services de données est conditionnée par une politique de confidentialité qui est composée d'un ensemble de 47 règles de confidentialité relatives à la protection des données personnelles. Pour chaque patient, nous avons généré aléatoirement les préférences sur la divulgation de données personnelles par rapport à 10 acteurs médicaux (par exemple, chercheur, médecin, infirmière, etc.) et à des objectifs d'usage différents (par exemple, la recherche scientifique). Ces préférences sont stockées dans une base de données Oracle indépendante et accessible via 10 services Web, chacun donne un accès à des fichiers de préférences par rapport à un type particulier de données médicales (par exemple, les traitements en cours). L'algorithme de composition est implanté en Java et exécuté sur un processeur Intel Core Duo 2,53 Ghz avec 4 Go de Ram sous le système Windows 7. Nous avons mené une série d'expérimentations pour mesurer les coûts engendrés par l'application de la politique de confidentialité sur l'approche de composition. Nous avons examiné deux séries de requêtes mashup. La première série, Set-1, contenait des requêtes sur un patient donné, chacune avec une taille différente: Q_1: « retourner les *informations personnelles* de la patiente Alice » (1 nœud-classe dans le graphe requête), Q_2: « retourner les *informations personnelles*, *allergies* et *les traitements en cours* pour la patiente Alice (3 nœuds-classe), et

Q_3: « retourner les *informations personnelles, allergies*, *les traitements en cours*, *l'état cardiaque*, et *le résultat des tests biologiques* du patient Alice » (5 nœuds-classe). La deuxième série, Set-2, utilise les mêmes requêtes Q1, Q2 et Q3, mais pour tous les patients vivant à Lyon. Ces trois requêtes sont formulées par un même acteur (chercheur) et pour le même objectif d'usage (recherche médicale). Les résultats obtenus sont illustrés dans la figure 9. Le temps indiqué comprend le temps de construction du mashup et le temps de d'exécution effectif. Dans Set-1, le temps d'exécution du mashup est négligeable (car le mashup est exécuté pour 1 seul patient) et l'application de la politique de confidentialité induit une légère augmentation impactée sur le temps de réécriture de la requête. Cela est dû au fait que le nombre de services utilisés pour récupérer les préférences de confidentialité (10 services) est largement faible par rapport au nombre de services utilisés pour récupérer des données (411 services). Dans le Set-2, le temps d'exécution s'accroit par un facteur égal au nombre de patients retourné. En revanche, l'application de la politique de confidentialité a un impact relativement faible sur le temps d'exécution.

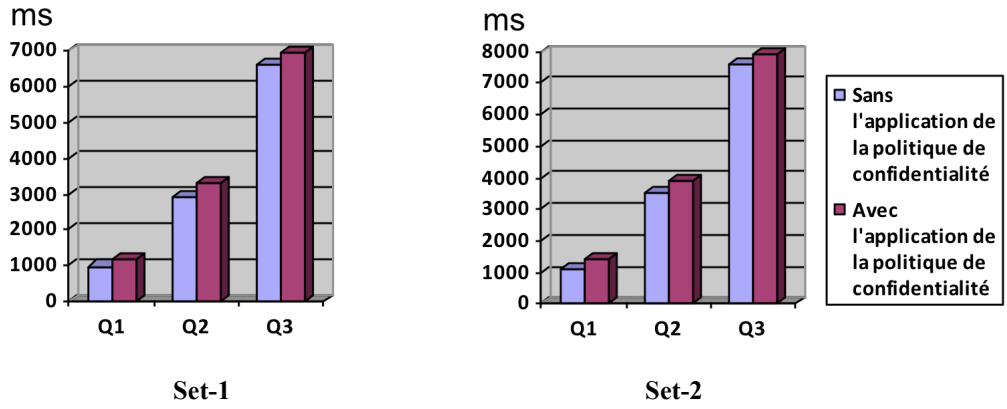


Figure 9. Résultats d'expérimentation

6. Conclusion

Dans cet article, nous avons présenté une approche déclarative de construction de mashup dans le cadre des services Daas qui prend en compte la protection des données personnelles lors de la composition des services pour le mashup. Ces services de données sont modélisés à l'aide des vues RDF paramétrées sur des ontologies de domaine; les vues définies sont ensuite utilisées pour annoter les fichiers de description WSDL (cette solution peut être appliquée aux services Web SaaS). L'approche de composition est basée sur une technique de réécriture des vues RDF. Ainsi, la requête utilisateur est réécrite en termes de services. La protection des données personnelles est prise en compte ensuite lors de la construction du mashup par l'application des contraintes de confidentialité comme des filtres lors de l'exécution de la composition. Ces contraintes désignent les règles d'usage concernant les données personnelles et sont spécifiées via une politique de confidentialité. Nous avons également présenté un prototype de composition et quelques résultats d'expérimentation.

Une des perspectives de notre travail consiste à améliorer l'approche de composition selon plusieurs dimensions: la première vise à étudier les problèmes d'optimisation de l'approche de composition en s'appuyant sur les travaux dans [43]. Nous analyserons ensuite la validité de notre modèle de protection de données personnelles selon un modèle d'estimation d'attaques d'adversaires [40] [41]. La seconde sera consacrée pour étendre l'annotation des services DaaS pour intégrer les contraintes de confidentialité. Ainsi chaque

service DaaS sera doté lui-même par un mécanisme de protection des données personnelles. Nous avons déjà proposé une première approche [18] qui permet d'identifier les différents points d'extension au niveau du fichier WSDL pour intégrer les contraintes de confidentialité. L'autre perspective concernera l'extension de l'approche de composition avec des services DaaS annotés.

7. Références

[1] H. L. Truong and S. Dustdar, "On analyzing and specifying concerns for data as a service," in APSCC, 2009, pp. 87–94.

[2] M. J. Carey, "Declarative data services: This is your data on soa," in IEEE International Conference on Service-Oriented Computing and Applications, SOCA 2007, 2007, p. 4.

[3] D. Butler, "Mashups mix data into global service," in Nature, January 2006.

[4] A. Jhingran, "Enterprise information mashups: Integrating in- formation, simply," in VLDB, 2006, pp. 3–4.

[5] S. S. Bhowmick, L. Gruenwald, M. Iwaihara, and S. Chatvichienchai, "Private-iye: A framework for privacy preserving data integration," in ICDE Workshops, 2006, p. 91.

[6] Yahoo inc. yahoo pipes. [Online]. Available: http://pipes.yahoo.com/pipes/

[7] Google inc. google mashup editor. [Online]. Available: http://code.google.com/gme/

[8] Intel. intel mash maker. [Online]. Available: http://mashmaker.intel.com/web/

[9] J. Tatemura, S. Chen, F. Liao, O. Po, and D. Agrawal, "Uqbe: uncertain query by example for web service mashup," in SIGMOD Conference, 2008, pp. 175–180.

[10] J. Tatemura, A. Sawires, O. Po, S. Chen, D. Agrawal, and M. Goveas, "Mashup feeds: : continuous queries over web services," in SIGMOD Conference, 2007, pp. 128–130.

[11] A.H.H.Ngu,M.P.Carlson,Q.Z.Sheng,andH.youngPaik, "Semantic-based mashup of composite applications," IEEE T. Services Computing, vol. 3, no. 1, pp. 2–15, 2010.

[12] T. Weise, S. Bleul, D. E. Comes, and K. Geihs, "Different approaches to semantic web service composition," in Third International Conference on Internet and Web Applications and Services, ICIW 2008, 2008, pp. 90–96.

[13] Q. Yu, X. Liu, A. Bouguettaya, and B. Medjahed, "Deploying and managing web services: issues, solutions, and directions," VLDB J., vol. 17, no. 3, pp. 537–572, 2008.

[14] M. A. Eid, A. Alamri, and A. El-Saddik, "A reference model for dynamic web service composition systems," IJWGS, vol. 4, no. 2, pp. 149–168, 2008.

[15] D. Martin, M. Paolucci, and M. Wagner, "Bringing semantic annotations to web services: Owl-s from the sawsdl perspec- tive," in ISWC/ASWC, 2007, pp. 340–352.

[16] M. Steinbrunn, G. Moerkotte, and A. Kemper, "Heuristic and randomized optimization for the join ordering problem," VLDB J., vol. 6, no. 3, pp. 191–208, 1997.

[17] M. Barhamgi, D. Benslimane, and B. Medjahed, "A query rewriting approach for web service composition," In IEEE T. Services Computing, vol. 3, no. 3, pp. 206–222, 2010.

[18] M. Mrissa, S-E. Tbahriti and H.L. Truong, "Privacy model and annotation for DaaS," In Proc. of European Conference on Web Services (ECOWS), Antonio Brogi, Cesare Pautasso, George Angelos Papadopoulos ed. Ayia Napa, Cyprus. pp. 3-10. 2010.

[19] http://www.w3.org/TR/P3P/

[20] Massimo Paoucci, Naveen Srinivasan, Grit Denker, Tim Finin and Katia Sycara Lalana Kagal, "Authorization and Privacy for Semantic Web Services," In IEEE Intelligent Systems (Special Issue on Semantic Web Services), vol. 19, no. 4, pp. 50-56. 2004.

[21] M. Ouzzani, A. Bouguettaya, B. Medjahed, and A. Rezgui, "Preserving Privacy in Web Services," Proceedings of the fourth international Workshop On Web Information And Data Management, 2002.

[22] OASIS 2005. (2005, February) OASIS "eXtensible Access Control Markup Language (XACML)," version 2.0. [Online]. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.

[23] http://www.zurich.ibm.com/security/enterprise-privacy/epal/Specification/index.html

[24] N. Li, T. Li, and S. Venkatasubramanian "t-Closeness: Privacy Beyond k-Anonymity and l-Diversity," Proc. of the International conférence of Data engineering (ICDE), 2007.

[25] K. Nissim, and C. Dwork, "Privacy-preserving data-mining on vertically partitioned databases," In CRYPTO, Lecture Notes in Computer Science, Ed. Santa Barbara, California, USA: Springer, August 2004, vol. 3152, pp. 528-544.

[26] U.S. Department of Health and Human Services Office for Civil Rights, "HIPAA administrative simplification regulation text," 2006.

[27] European Commission, “Directive 95/46/ec of the european parliament and of the council of 24 october 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data,” 2005.
[28] S. De Capitani di Vimercat, S. Foresti, S. Jajodia, and P. Samarati, “Access Control Policies and Languages in Open Environments,” in Secure Data Management in Decentralized Systems, Springer US, Ed., 2007, vol. 33, pp. 21-58.
[29] R. S. Sandhu and P. Samarati, “Authentication, Access Controls, and Intrusion Detection,” IEEE Communications, pp. 1928-1948, 1997.
[30] L.F. Cranor, “Special Issue on Internet Privacy,” In Journal of Comm. ACM, vol. 42, no. 2, pp. 28-38, 1999.
[31] Harris IBM, “The IBM-Harris Multi-National Consumer Privacy Survey,” *Privacy & American Business*, vol. 7, no. 6, 2000.
[32] A. Y. Halevy, “Answering queries using views: A survey”. 10, pp. 270-294, 2001.
[33] http://www.w3.org/TR/rdf-sparql-query/
[34] U. Srivastava, K. Munagala, J. Widom, & R. Motwani, “Query Optimization over Web Services, ” VLDB, pp. 355-366, Seoul, Korea. 2006.
[35] M. Sabesan, & T. Risch, “Adaptive Parallelization of Queries over Dependent Web Service Calls,” 1st IEEE Workshop on Information & Software as Services, WISS 2009. Shanghai, China.
[36] M. Sabesan, & T. Risch, “Querying Mediated Web Services,”. 8th International Information Technology Conference, IITC 2006. Colombo, Sri Lanka.
[37] M. Carey, “Declarative data services: This is your data on SOA,”. In Proceeding of the IEEE International Conference on Service-Oriented Computing and Applications, Washington, DC, USA, 2007.
[38] R.D. Hof, “Mix, match, and mutate,” Business Week, July 2005.
[39] A. H. H. Ngu, M. P. Carlson, Q. Z. Sheng, and H.-y. Paik. “Semantic-based mashup of composite applications,”. IEEE Trans. Serv. Comput., 3:2–15, January 2010.
[40] D. Kifer, “Attacks on privacy and de finites theorem,” Proc. of the SIGMOD, 2009.
[41] A. Machanavajjhala, J. Gehrke, and M. Götz, “Data Publishing against Realistic Adversaries,” Proc. of the VLDB, 2009.
[42] http://chicago.everyblock.com/crime/
[43] Q. Yu, and A. Bouguettaya, “Framework for Web service query algebra and optimization,” TWEB, 2, 1, 2008.