Une approche algébrique pour la réutilisation et l'orchestration de services dans les systèmes d'information

Yann Pollet

Laboratoire CEDRIC, Conservatoire National des Arts et Métiers
Chaire d'Intégration des Systèmes
292, rue Saint-Martin
F-75003 Paris

yann.pollet@cnam.fr

RÉSUMÉ. *Dans les systèmes d'information, il est aujourd'hui essentiel de mettre en œuvre rapidement et sans développements lourds de nouvelles fonctionnalités en réponse aux évolutions métiers, en réutilisant au mieux les services déjà présents. Dans le contexte de l'accès à des informations dispersées au sein d'une fédération de systèmes indépendants, nous adressons ici le problème de découverte et de l'orchestration automatisées des services. Nous proposons dans ce cadre une approche basée sur une ontologie de domaine et la capture de la sémantique des services existants par le biais d'expressions algébriques opérant sur des propriétés de cette ontologie. Un algorithme permet alors de générer des plans d'orchestration de services dotés de propriétés d'optimalité vis-à-vis de la QoS. Un prototype et l'évaluation sur le cas d'un système d'information de santé a permis la validation de cette approche.*

ABSTRACT. *In order to deal with business evolutions, it is important today to enable fast developments of new functionalities on the basis of existing Services reuse. In the context of information search within a federation of independent systems, we address the problem of automated discovery and orchestration of Services. We propose an approach based on a domain ontology, and on the Services semantics capture by the mean of ontology properties and algebraic expressions. We present an algorithm that generates orchestration plans, with characteristics of optimality regarding QoS. The approach has been validated by a prototype and an evaluation in the case of an Health Information System.*

MOTS-CLÉS : *ontologie, web sémantique, annotation sémantique, algèbre de description de service, composition de services, découverte de service.*

KEYWORDS: *ontology, semantic web, semantic annotation, service description algebra, service composition, service discovery.*

DOI:10.3166/ISI.15.5.63-88

1. Introduction

L'évolutivité des systèmes d'information informatisés est aujourd'hui au cœur des démarches de rationalisation des développements informatiques des organisations. Il est impératif de rendre le système d'information apte à supporter, sans développements majeurs, et dans des délais très courts, de nouvelles fonctionnalités induites par les évolutions métiers.

Ce défi se heurte à d'importantes difficultés, en particulier dans le cas ici considéré, de situations où coexistent plusieurs systèmes conçus de manière indépendante, autonomes et technologiquement hétérogènes (tel par exemple le cas de compagnies issues de fusions, ou encore celui d'entités juridiquement et économiquement distinctes, fonctionnant en réseau). Dans l'évolution du système global, une difficulté majeure est celle de la réutilisation adéquate des services logiciels déjà développés pour répondre à un nouveau besoin, et dans le respect d'une qualité de service (QoS) requise.

Nous considérons ici le cadre particulier des activités dites *expertes* (*artful activities*), telles l'ingénierie de projets ou le cadre de la santé, où la décision humaine joue un rôle majeur, et où un besoin essentiel est alors celui de l'accès à des informations multiples dispersées dans différents systèmes ou applications locales. Dans ces activités, permettre un accès aisé à ces informations par simple réutilisation et composition de services logiciels déjà existants est un problème majeur de l'informatisation. Cela nécessite, pour tout nouveau problème, d'identifier les services existants adéquats et de les composer convenablement pour produire la réponse attendue. Ceci sous-entend, à la base, la capacité d'indexer préalablement les services existants sur le plan de leur *sens*, c'est-à-dire ce qu'ils produisent et à partir de quoi. Le nœud du problème est donc ici la capture de la sémantique associée à un service d'accès à l'information. L'architecture orientée service (SOA), qui préconise le Service comme composant logiciel structurant, est ici la base requise, mais n'a pas pour but en elle-même de répondre à ce problème.

Dans cet article, nous proposons une approche basée sur une *formalisation algébrique* opérant sur les concepts d'une *ontologie du domaine*, et permettant une indexation sémantique des Services. La capture de sémantique par ce biais permet en particulier de proposer une *génération automatisée d'orchestrations de services* en réponse à un nouveau besoin d'accès à l'information.

La suite de l'article est organisée de la manière suivante : la section 2 présente le contexte de la recherche et l'état de l'art. La section 3 présente un exemple du *domaine de la Santé*, ainsi que les hypothèses adoptées. Dans la section 4, l'*algèbre RCS*, développée comme fondation mathématique, est présentée. La section 5 décrit le *modèle d'indexation* de Services associé, et la section 6 présente son utilisation pour la *génération automatisée* optimale de *plans d'orchestration de Services*. Enfin la section 7 donne une brève présentation d'un prototype développé sur la base de notre approche et la section 8 conclut.

2. Contexte de la recherche

Les approches en rapport avec notre problématique sont, de manière générale, celles ayant trait à la *sémantisation* des services (McIlraith *et al*., 2001). Plus précisément, afin de permettre leur réutilisation, les services doivent tout d'abord être décrits dans un langage commun d'inter-opération. Sur cette base, il convient de fournir des mécanismes de découverte permettant de rechercher des services conformes à un besoin donné. Il faut enfin composer les services découverts afin de réaliser un service de plus haut niveau conforme au nouveau besoin applicatif.

Description de services. Les langages de description de services web fournissent des primitives permettant de décrire les propriétés pertinentes d'un service. Ceci va d'une description purement syntaxique, telle que celle offerte par le langage WSDL (*Web Service Description Language*), à une description sémantique intégrant le *sens*, c'est-à-dire de ce que fait le service.

Les approches les plus représentatives sont OWL-S (Martin *et al.*, 2007), WSMF (Fensel *et al*., 2002), WSMO (Roman *et al.*, 2005), SAWSDL (Kopeck *et al*., 2007), METEOR-S, et IRS-II. Elles ont comme objectif commun la découverte automatique de services satisfaisant un besoin spécifique. OWL-S est une ontologie générique de description de services web, basée sur les standards OWL et WSDL. WSMF est un cadre générique de réalisation de services, et offre un cadre méthodologique plus qu'une infrastructure opérationnelle. WSMO est une approche basée sur des fondements conceptuels de WSMF pour la réalisation des services sémantiques. METEOR-S propose un enrichissement sémantique basé sur l'annotation sémantique de fichiers WSDL. Enfin, IRS-II est une approche comparable liée au framework UPML.

L'extension de WSDL *via* des annotations sémantiques semble être la voie la plus naturelle car conforme à la standardisation. On peut cependant constater qu'aucune de ces approches ne permet de formaliser ce qu'est vraiment le *sens* d'un service, ce qui exclut dans la pratique toute sélection automatisée de service par un agent logiciel sans assistance de l'usager, au moins pour validation.

Découverte de services. On peut distinguer sur ce plan les approches syntactiques pour la découverte de services web des approches dites sémantiques. S'inspirant du modèle classique de recherche d'information, les approches syntactiques permettent d'interroger un catalogue de services web par le biais de requêtes basées sur des mots-clés. On peut retenir ici les annuaires UDDI ainsi que des portails de services web comme QWSdataset et le moteur de recherche Seekda. Certaines approches sémantiques proposent d'étendre le standard UDDI, en insérant une couche sémantique. D'autres approches adoptent des techniques de recherche d'informations (RI) pour calculer des similitudes entre descriptions de services, des techniques de raisonnement opérant sur les descriptions sémantiques ou encore des approches hybrides combinant techniques de RI et raisonnement. Il faut ici remarquer que le principe de conformité à une signature de fonction, ou à des mots-

clés, ou encore celui de matching approché, ne sont aucunement en mesure d'assurer qu'un service répond effectivement bien à un besoin bien formulé.

Composition de services. La composition peut être vue comme un mécanisme permettant l'intégration de services existants pour réaliser une nouvelle application. Pour passer d'un besoin à une composition convenablement structurée, il est nécessaire de suivre des étapes qui vont de la spécification a une composition concrète exécutable : 1) la définition de l'architecture fonctionnelle, qui identifie les fonctionnalités attendues de l'application ; 2) l'identification des services (découverte), où on détermine les services pertinents ; 3) la sélection des services où, à partir des services identifiés, on sélectionne ceux répondant correctement aux besoins ; 4) la médiation, où des techniques de médiation sont mises en œuvre afin que les services sélectionnés interprètent correctement les données échangées ; et enfin 5) l'invocation des services. La composition de services a été abordée par un nombre considérable de travaux. En dépit de tous les efforts déployés, la composition de services reste un problème très complexe et non résolu de manière satisfaisante. Cette complexité tient en partie au fait que des approches rigoureuses de composition de services ne peuvent être envisagées sans réelle capture du sens.

Ces dernières années, plusieurs communautés se sont intéressées à la problématique de composition de services, et des solutions ont été proposées dans les communautés des bases de données, du web sémantique, et plus récemment dans celles de l'intelligence artificielle. Les solutions proposées peuvent être classifiées suivant deux axes : 1) le degré de participation de l'utilisateur à la définition du schéma de composition (manuelle, semi-automatique ou automatique), et 2) selon que la sélection des services et la gestion du flot sont faites ou non *a priori* (approches dite statiques ou dynamiques). Parmi les approches les plus représentatives actuellement développées, on peut citer :

– *les langages de définition et d'exécution de workflows*, avec BPML (*Business Process Modeling Language*), WSCL (*Web Service Conversation Language*) (Banerji *et al*., 2002), WSCI (*Web Service Choreography Interface*), WSFL (*Web Services Flow Language*), XLANG, BPEL4WS (*Business Process Execution Language For Web Services*) (Andrews *et al*., 2003), ainsi que YAWL, XPDL et WS-CDL ;

– *la composition de services à base de workflows*. La plate-forme eflow (Casati *et al*., 2000) a été proposée pour la spécification, la création et la gestion de services composites. Laukkanen *et al.* (2003) identifient pour leur part différentes solutions possibles pour composer dynamiquement des processus métiers. Enfin, Osman *et al.*, (2005) proposent de limiter les points négatifs des techniques de composition de type workflow en utilisant les technologies du web sémantique ;

– *la composition de services basée sur des techniques de raisonnement*. L'IA et les approches basées sur des techniques de planification sont généralement considérées comme les plus prometteuses. On peut distinguer la composition de services 1) à base de logique (calcul situationnel), 2) à base de planification, en

particulier avec PDDLP et des réseaux de type HTN, 3) à base de règles, 4) par synthèse de programmes, et enfin 5) la composition orientée QoS.

Force est de constater qu'aucune de ces approches n'est suffisamment mûre pour envisager une réelle automatisation. En nous limitant au domaine particulier des services *sans effet de bord*, nous nous inscrirons ici dans une démarche à base de planification, mais sur la base d'une formalisation algébrique rigoureuse du *sens*, reposant sur une ontologie du domaine. Sur cette base, nous montrerons qu'il est possible d'opérer des générations automatisées pertinentes de plans d'orchestration.

3. Cas d'étude et hypothèses

3.1. *Un exemple : la fédération de systèmes d'information de santé*

Pour illustrer notre problématique, nous considérons ici le domaine des systèmes d'information de santé. Dans une aire régionale, les différents établissements sont dotés chacun de leur propre système, coopérant dans le cadre du processus global de soin. Un dossier médical est un ensemble d' « événements médicaux » datés (diagnostics, traitements, actes), et gérés par une institution donnée (un patient pouvant avoir un ou plusieurs dossiers par établissement). Les praticiens disposent de droits d'accès à un dossier, en fonction de leurs rôles dans le processus de soin du patient. Outre les systèmes d'information des établissements, des serveurs d'identités peuvent synthétiser au niveau régional les informations administratives de patients et référencer les différents dossiers existant pour un même patient.

La figure 1 illustre un fragment significatif d'une ontologie simplifiée du domaine (où, par souci de clarté, nous n'avons pas fait figurer les attributs des classes).

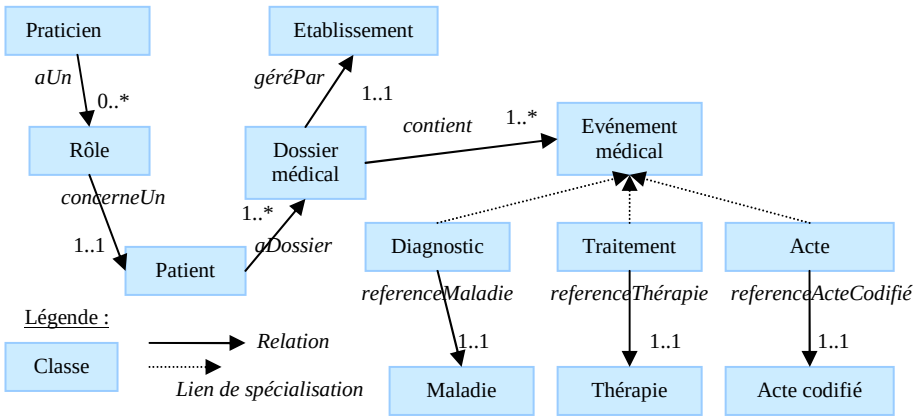


Figure 1. *Fragment simplifié de l'ontologie du domaine*

Le cloisonnement en systèmes indépendants s'oppose à la globalité des processus de soin du patient, et la décision médicale induit un besoin d'accès à des informations dispersées au sein de différents systèmes. Un exemple est la décision thérapeutique, qui requiert l'accès à un dossier patient complet, et qui conduit en général à la sollicitation de différents systèmes. Il existe de nombreux autres exemples tels que les études épidémiologiques ou l'accès à des cas anonymes.

3.2. *Hypothèses*

Conformément à une architecture orientée service, on suppose que chaque établissement n'expose pas directement ses données mais fournit des accès (dans notre cas par le biais d'un réseau sécurisé) à des informations bien identifiées, et sous la forme de Services. Nous considérons ici les *services d'accès à des données*, à l'exclusion de services ayant trait à la création ou mise à jour de données, ou encore de services ayant un effet sur le monde réel. Les services considérés sont donc des services *sans état* et *sans effets de bord*. On ne fait pas ici d'hypothèses sur les technologies mises en œuvre, ces services pouvant être implémentés par des web services, ou d'autres solutions technologiques d'appel de fonctions à distance. Enfin, les technologies de médiation sont hors du champ de notre recherche, et on suppose développés les médiateurs convenables permettant une présentation homogène des services conformes aux classes et propriétés de l'ontologie.

3.3. *La notion de Service*

Nous considérons ici un ensemble de Services $\{S_{i,}\}$ fournis par différents systèmes participants, ainsi qu'une ontologie du domaine O. Celle-ci définit un ensemble de classes $\{C_{i,}\}$, dotées de propriétés attributs $\{V_{i,j}\}$ et de relations directionnelles $\{R_{i,k}\}$ entre classes. Dans un premier temps, nous définissons simplement un *Service* comme une fonction accessible à distance et délivrant en sortie un ensemble résultat sur la base de paramètres fournis en entrée soit :

$\{o\} = S\ (o_1, \ldots, o_n)$, ou bien $\{v\} = S\ (o_1, \ldots, o_n)$

où o et les o_i sont des individus dont les types correspondent à des classes de l'ontologie O, et où v est une donnée de type correspondant à un attribut d'une classe. Un Service encapsule le détail des opérations exécutées, et peut, de manière transparente, extraire l'information requise d'une base de donnée locale, procéder à un traitement (par exemple l'application de règles dans la détermination des droits d'accès et de leurs durées), ou encore appeler des fonctions internes au système.

4. L'algèbre RCS

4.1. *Principes généraux*

Le principe à la base de notre approche consiste à définir la sémantique d'un Service par la mise en correspondance de celui-ci avec une propriété de l'ontologie, relation ou attribut. Le cas le plus simple est celui d'un Service dont l'invocation avec l'individu o en entrée délivre en sortie un objet ou un ensemble {o'} d'objets en *relation avec un objet donné en entrée* via *une relation donnée R*. Par exemple, si un Service délivre le contenu d'un dossier médical à partir de l'identifiant de ce dossier, le sens du Service est parfaitement défini par la Relation *Contient* de l'ontologie. Le principe consiste donc à caractériser la sémantique d'un Service par la propriété éventuelle de l'ontologie que celui-ci réalise.

Mais ce principe doit être étendu à des cas plus complexes. En effet, dans la pratique, rien n'oblige un Service à implémenter exactement une relation ontologique. En particulier un Service peut correspondre à une propriété non définie explicitement dans l'ontologie, mais définie comme un chemin dans le graphe des relations (par exemple un Service délivrant le contenu complet du dossier patient à partir de l'identification de ce patient correspond à un chemin combinant les deux relations *aDossier* et *Contient*). Par ailleurs, un Service peut ne réaliser que partiellement une propriété. Ainsi, si un service S afférent à un établissement E délivre, pour un patient donné, l'ensemble des événements médicaux $\{E_i\}$ survenus dans cet établissement, l'invocation du Service ne délivre qu'une partie des événements médicaux correspondant à la relation précédente.

Il nous faut donc étendre les propriétés *natives* de l'ontologie par de nouvelles propriétés que nous appellerons *propriétés dérivées* de celles-ci. Un besoin d'accès à une information pourra être défini comme le problème d'évaluation d'une propriété *native* ou *dérivée*. Une telle propriété définit une *requête globale*, qu'il conviendra de transformer en invocations de Services convenables.

Il pourrait sembler que le recours à une ontologie de domaine comme base obligée est une hypothèse forte, et donc limitative, car restreignant la réutilisation des services au seul domaine couvert par cette ontologie. Cette approche nous semble au contraire tout à fait justifiée dans le cas de services métiers. En premier lieu, il serait tout à fait illusoire de prétendre à une sélection et une composition automatisées de Services sans une quelconque capture préalable des concepts utilisés par le métier. L'ontologie joue ce rôle. En second lieu, la portée de cette approche, dans chaque domaine d'application, ne sera véritablement limitée que par la richesse de l'ontologie utilisée. Enfin, dans de nombreux domaines, comme celui de la santé ici évoqué, où les concepts sont bien formalisés par le biais de standards, la portée de la réutilisation s'étendra, non à un domaine fonctionnel particulier, mais bien à toute la large classe des systèmes reposant sur les concepts décrits.

4.2. *Classes dérivées*

A partir des classes de l'ontologie, appelées ici *classes natives*, on peut définir de nouvelles classes *dérivées*, par application des opérateurs OWL suivants :

Intersection : $C = C_1.C_2$ ($C_1 \cap C_2$), définie par : $x \in C_1.C_2$ ssi $x \in C_1$ ET $x \in C_2$,

Union : $C = C_1 + C_2$ ($C_1 \cup C_2$), définie par : $x \in C_1 + C_2$ ssi $x \in C_1$ OU $x \in C_2$

où C_1 et C_2 sont des *classes natives,* ou encore des *classes dérivées* déjà définies.

Les propriétés (attributs, relations) applicables à un $x \in C_1.C_2$ sont les propriétés applicables à la fois aux $x \in C_1$ et aux $x \in C_2$. Les propriétés applicables à un $x \in C_1 + C_2$ sont la réunion des propriétés applicables aux $x \in C_1$ et $x \in C_2$

Restriction de propriétés : $C' = C^P$, définie par $x \in C^P$ ssi $x \in C$ ET P(x)

où C est un *classe native* ou *dérivée* déjà définie, et P un prédicat défini par une expression logique ET-OU de prédicats élémentaires de la forme $V_i \Theta v$, où V_i est une propriété définie sur C, v une valeur de celle-ci, et où enfin Θ est un opérateur de comparaison ayant un sens sur le domaine de valeur concerné. Les propriétés applicables à un $x \in C^P$ sont celles applicables aux $x \in C$

Propriété (treillis ontologique). *Etant donné une ontologie O, l'ensemble des classes natives de O, complété par l'ensemble des classes dérivées de O générées par intersection, union, et restriction de propriétés est un treillis.*

La relation d'ordre associée notée « $\leq$ » est définie par : $C_1 \leq C_2$ ssi $C_1 \subset C_2$ (inclusion ensembliste), et toute propriété de C_2 est aussi propriété de C_1. Cette relation étend la spécialisation/généralisation. Une classe *native* ou *dérivée* sera dite dans la suite classe *étendue*.

4.3. *Opérations algébriques sur les relations*

Nous définissons maintenant un ensemble d'opérateurs algébriques permettant de dériver de relations déjà existantes de nouvelles relations, dites *dérivées*, et attachées à des classes *natives* ou *dérivées*. Une relation *native* ou *dérivée* sera dite relation *étendue*. Nous définissions deux opérateurs sur l'ensemble $\mathcal{R}$ des relations : la *composition* et *l'union* de relations, ainsi qu'un constructeur de relations dites *filtres*.

On écrira <R, x, y> pour exprimer le fait que les individus x et y sont en relation par R. Par ailleurs, dom (R) et range (R) noterons respectivement le *domaine* de la relation R (espace des x) et l'ensemble d'arrivée (espace des y). Enfin, minCard (R) et maxCard (R) noterons respectivement les cardinalités minimum et maximum de R (qui vaudront 0 et ∞ par défaut).

4.3.1. *Composition*

Définition (opérateur de composition). *Pour toutes relations R_1 et R_2, $R_1 * R_2$ est la relation telle que dom ($R_1 * R_2$) = dom (R_1), range ($R_1 * R_2$) = range (R_2), et définie par <R, x, z> ssi il existe y ϵ range (R_1) ∩ dom (R_2) tel que* ***<R_1, x, y> et <R_2, y, z>***

Ici, rien n'exige que range (R_1) = dom (R_2), et le résultat $R = R_1 * R_2$ est toujours défini. Si range (R_1) ∩ dom (R_2) = Ø, alors R est une relation *nulle*, où aucun individu n'a de correspondant. L'opérateur de composition est une *loi de composition interne* $\mathcal{R}$. Il est *associatif* mais pas commutatif. On a par ailleurs : minCard ($R_1 * R_2$) = minCard (R_1) . minCard (R_2), et maxCard ($R_1 * R_2$) = maxCard (R_1) . maxCard (R_2).

La figure 2 illustre l'opérateur de composition ainsi défini.

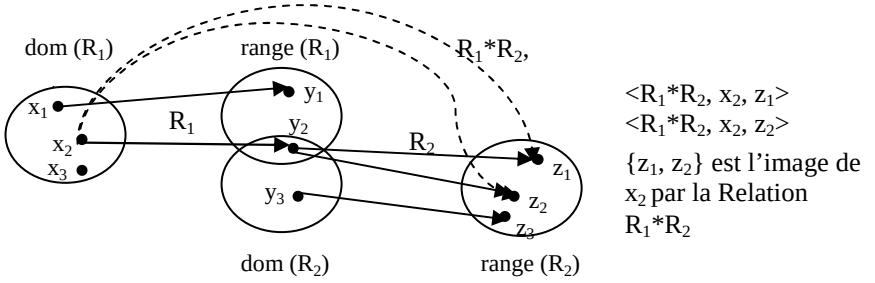


Figure 2. *Illustration de l'opérateur de composition*

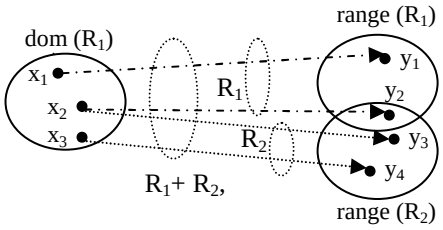
Dans notre cas d'étude illustré par la figure 1, une nouvelle relation *Antecedent* (pathologies dont le patient a déjà souffert), attachée à la classe *Patient*, peut être par exemple définie par la composition *aDossier * contient * referenceMaladie*.

4.3.2. *Union*

Définition (opérateur d'union). *Soit deux relations R_1 et R_2, l'union $R = R_1 + R_2$ est la relation telle que dom (R1 + R2) = dom (R1) ∩ dom (R_2), range (R1 + R2) = range (R1) U range (R2), et <R, x, y> ssi* ***<R1 x, y> ou <R2, x, y>***

La relation R est une propriété de la classe dom (R_1) ∩ dom (R_2), à valeurs dans range (R_1) U range (R_2). L'image d'un individu x par R sera l'union ensembliste des images de x par R_1 et par R_2. $R_1 + R_2$ est toujours défini, (éventuellement comme une relation *vide*, en particulier dans le cas où dom (R_1) ∩ dom (R_2) = Ø) et l'union est donc une *loi de composition interne* sur $\mathcal{R}$. Elle est *commutative*, *associative* et la composition est *distributive* par rapport à l'union. On a par ailleurs minCard (R_1 + R_2) = Max (minCard (R_1), minCard (R_2)), et maxCard ($R_1 + R_2$) = Min (maxCard (R_1), maxCard (R_2)).

La figure 3 illustre l'opérateur d'union ainsi défini.



$<R_1+ R_2, x_1, y_1>$
$<R_1+ R_2, x_2, y_2>$
$<R_1+ R_2, x_2, y_3>$
$<R_1+ R_2, x_3, y_4>$
y_1 est l'image de x_1
$\{y_2, y_3\}$ est l'image de x_2
y_4 est l'image de x_3

Figure 3. *Illustration de l'opérateur d'union*

4.3.3. *Filtre de restriction*

Nous pouvons souhaiter définir une nouvelle relation qui associe à tout patient les événements médicaux correspondant à des pathologies particulières, ou encore accéder seulement aux actes ayant été exécutés dans certains établissements. Pour définir des telles notions, il faut pouvoir restreindre une relation existante avec des contraintes restreignant les individus de départ et/ou d'arrivée. Pour ce faire, nous introduisons le constructeur de relation appelé *filtre*, puis l'opérateur de *restriction*.

Définition (filtre). *Pour toutes classes C_1 et C_2, on définit la relation $[C_1, C_2]$, appelée filtre de C_1 vers C_2, comme la relation telle que dom $([C_1, C_2]) = C_1$, range $([C_1, C_2]) = C_2$, et $< [C_1, C_2]$, o, o> ssi* $\boldsymbol{o \in C_1 \cap C_2}$

$[C_1, C_2]$ associe à tout individu de C_1 soit l'individu lui-même, soit un ensemble de valeurs vide. Un individu y de C_2 ne peut avoir d'antécédent par $[C_1, C_2]$ que si $y \in C_1$. Si $C_1 = C_2 =$ Universal $= \cup C_i$, alors $[C_1, C_2]$ est simplement la relation identité. Si $C_2 = C_1^P$, où P est un prédicat défini sur C_1, alors $[C_1, C_1^P]$ est une relation de sélection selon le prédicat P. Nous avons en particulier la propriété $[C, C^{PRE1}] * [C, C^{PRE2}] = [C, C^{PRE1 \text{ AND } PRE2}]$.

Nous définissons alors la *restriction* d'une relation de la manière suivante.

Définition (restriction). *Soit une relation R, deux prédicats PRE (x) et POST (y) s'appliquant respectivement aux individus x de la classe dom (R) et à ceux y de la classe range (R), la restriction de R selon PRE et POST est définie par :*

$$R^{PRE, POST} = [C_1, C_1^{PRE}] * R * [C_2^{PRE}, C_2] \text{ où } C_1 = dom\ (R) \text{ et } C_2 = range\ (R)$$

La relation $R^{PRE, POST}$ associe donc à tout individu x de C_1 vérifiant PRE (x) son image y par R, à la condition qu'elle vérifie POST (y), et une image vide sinon. La figure 4 illustre l'opérateur de restriction.

Par exemple, dans notre cas illustratif, nous pouvons définir à partir de la relation *contient* qui associe à un dossier médical son contenu, de nouvelles relations dotées de sens plus spécifiques : 1) Relation associant aux seuls dossiers de l'établissement H leur contenu (et donnant une valeur vide pour les autres dossiers) : $R_1 = \text{Contient}^{\text{GéréParH, True}}$, où

GéréParH (d)= d.GéréPar = « H ». 2). Relation associant à tout dossier l'ensemble des diagnostics : R_2 = Contient $^{True,\ EstUnDiagnostic}$, où EstUnDiagnostic = Is (Diagnostic).

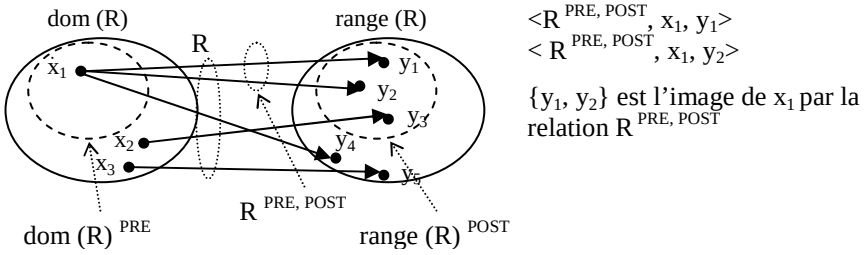


Figure 4. *Illustration de la restriction*

4.4. *Décomposition canonique d'une relation*

Considérons une relation R, deux ensembles de prédicats $\{PRE_i ; i=1, ..,n\}$ et $\{POST_j ; j=1, ..,m\}$ s'appliquant respectivement aux classes dom (R) et range (R), et tels que :

PRE_1 OU …. OU PRE_n = True, et $POST_1$ OU …. OU $POST_m$ = True,

où True est le prédicat toujours vrai. Nous avons immédiatement :

$R = \sum_{i=1,..,n \text{ OU } PREi = True} R^{PREi,\ True}$ et $R = \sum_{j=1, \ldots,m \text{ OU } POSTj = True} R^{True,\ POSTj}$

et plus généralement : $R = \sum_{i=1,..,n,\ j=1, \ldots,m \text{ OR } PREi = True, \text{ OR } POSTj = True} R^{PREi,\ POSTj}$

Lorsque R est une relation *native*, et lorsque les prédicats PRE_i et $POST_j$ sont mutuellement exclusifs, la propriété définit une décomposition canonique de R.

4.5. *Expressions algébriques et relations étendues*

Les opérateurs définis ci-avant permettent de formuler des expressions algébriques où les opérandes sont des relations *étendues*. Il est à noter qu'une telle expression est toujours définie quels que soient les domaines et ensemble de valeurs des relations opérandes, et toute formule valide définit une nouvelle *relation étendue*. Si l'une des opérandes est une relation *étendue*, une autre formule doit préalablement avoir défini celle-ci, et les définitions cycliques sont exclues. L'ensemble des relations *étendues* associé à une ontologie O peut alors être défini par une suite d'expressions algébriques de la forme :

$R_i = EXPR_i (R_{1,i}, …., R_{ni,\ i})$; 1=1, …, N

où $EXPR_i$ est une expression dont les opérandes $R_{k,\ i}$ k = 1, …, n_i sont soit des relations *natives* de l'ontologie soit des relations *dérivées* parmi les R_1, …, R_{i-1}

Exemples 1. Relation R_1 associant à tout patient enregistré dans l'établissement H l'ensemble de ses dossiers : R1 = aDossier * [DossierMédical, DossierMédical ^{enregistréDansH}] * contient, où enregistréDansH (d) = d.géréPar = « H ». R_1 est une propriété de la classe Patient. *Exemple 2.* Relation associant à tout patient ayant été diagnostiqué pour une hépatite d'origine virale les établissements où il a un dossier : aDossier * [DossierMédical, DossierMédical ^{contientHépatiteVirale}] * géréPar, où contientHépatiteVirale (d) = « Hépatite d'origine virale » IN d.contient

4.6. *Attributs étendus*

Pour définir de nouvelles propriétés attributs non plus seulement relations mais aussi attributs, nous utilisons l'opérateur de *projection* qui permet l'accès aux valeurs d'un attribut relatives à un individu donné.

Définition (projection). *Soit une classe native C, V un attribut de C, la projection C.V de la classe C sur l'attribut V est la fonction associant à tout individu x de C l'ensemble {V_i} des valeurs attachées à l'individu x par l'attribut V.*

Avec la projection, il est possible de définir de nouvelles propriétés attributs par des expressions de la forme V' = EXPR (R_1, …, R_n) . V, où EXPR (R_1, ., R_n) est une expression algébrique des relations, et V une propriété attribut de la classe C' = range (EXPR). Ceci définit un nouvel attribut attaché à la classe dom (EXPR), et tel que range (V') = range (V). Pour V' = R . V, on a MaxCard (V') = MaxCard (R) . MaxCard (V) et MinCard (V') = MinCard (R) . MinCard (V). Une telle expression définit un *attribut dérivé*. Un attribut *natif* ou *dérivé* sera dit *attribut étendu*.

5. Un modèle de correspondance entre services et propriétés ontologiques

5.1. *Principes de base*

Le principe de notre approche consiste à capturer *la sémantique d'un Service* par la notion de *relation*. Dans le cas le plus simple, la transformation opérée par un **Service (1, 1)** (une entrée, une sortie) correspond directement à une relation de l'ontologie (par exemple un Service prenant en entrée un patient et délivrant en sortie l'ensemble des identifiants de dossiers, correspond directement à la relation *aDossier*). On définit ainsi une notion de *correspondance* entre un Service et la relation que ce Service *réalise*. Nous appellerons *mapping*, un tel lien. Plus généralement, il est possible de définir le sens d'un Service par un *mapping* avec une relation *étendue* de l'ontologie, c'est-à-dire avec une expression algébrique d'autres propriétés ontologiques. Nous considérerons plus généralement ensuite le cas des Services de **type (n, p)**, (n entrées et p sorties) pour lequel il nous faudra disposer d'un modèle de correspondance plus général que la simple association directe. Enfin, afin d'être en mesure de capturer des *aspects non fonctionnels* (liés

par exemple aux performances ou à la disponibilité du Service), nous introduirons une modélisation de la *qualité de service*.

5.2. *Correspondances simples*

Nous considérons ici un **Service de type (1, 1),** dont l'entrée est un individu d'une classe C_{in} = In (S), et la sortie est un ensemble d'individus de la classe C_{out} = Out (S). L'espace de sortie de S est donc $2^{Out(S)} = 2^{Cout}$, soit : $S : C_{in} \rightarrow 2^{C_{out}}$. Le lien fonctionnel entre l'entrée et la sortie peut être capturé dans ce cas par le biais d'une correspondance avec une relation *native* ou *étendue* R de l'ontologie.

Définition (réalisation d'une relation par un service). *Soit un Service de type (1, 1), où In (S) et Out (S) sont des classes natives ou étendues de l'ontologie O, et soit R une relation native ou étendue de O. Nous dirons que le Service réalise la relation R ssi* $y \in S(x) \Leftrightarrow <R, x, y>$

On a donc dans ce cas In (S) = dom (R) et Out (S) = range (R). Nous écrirons : <MAP, S, R> (*assertion de correspondance*). Un service peut également *réaliser* un attribut. Considérons un service S de type (1, 1), dont l'espace d'entrée In (S) est une classe C_{in} de l'ontologie et dont l'espace de sortie est 2^D, où D est un datatype tel que Integer, String, Date…

Définition (réalisation d'un attribut par un service). *Soit un Service de type (1, 1) où In (S) est une classe native ou étendue de l'ontologie O, et Out (S) un datatype D. Soit V un attribut natif ou étendu de type D. Nous dirons que S réalise V ssi* $v \in S(x) \Leftrightarrow <V, x, v>$

Afin de simplifier la présentation, nous ne considérerons plus dans la suite que les Services *orientés individus* (délivrant des ensembles d'individus), à l'exclusion des Services *orientés datatypes* (délivrant des ensembles de valeurs) sachant que des développements similaires à ceux présentés peuvent être faits concernant les seconds.

5.3. *Correspondances restreintes*

Un Service peut ne réaliser qu'une partie d'une relation, c'est-à-dire ne s'appliquer qu'à un sous-domaine de cette relation, et/ou ne délivrer qu'une partie des résultats attendus. Dans ce cas, nous avons une propriété plus faible :

$\{y_i\} = S(x) \rightarrow <R, x, y_i>$ (l'implication inverse n'étant pas nécessairement vraie). Nous dirons que S est une *réalisation partielle* de R. Nous considérons ici le cas où la

partie de la relation R réalisée par R obéit à un **critère de rationalité**, lié aux règles de l'organisation, et peut donc être caractérisée par des critères bien identifiés[1].

Définition (assertion de correspondance). *Une assertion de correspondance est une assertion impliquant un Service S, une relation R, deux prédicats PRE et POST respectivement définis sur dom (R) et range (R) et exprimant le fait que S est une réalisation de* $R^{PRE, POST}$ [2] *soit :*

$y \in Y = S(x) \Leftrightarrow (<R, x, y> \text{ AND } PRE(x) \text{ AND } POST(y_i))$

Nous écrirons : <MAP, S, R, PRE, POST>, équivalent à <MAP, S, $R^{PRE, POST}$ >

5.4. *Une algèbre de Services*

A ce stade, il nous faut pouvoir définir de nouveaux Services, sur la base de Services existants, en opérant en particulier des combinaisons d'exécutions séquentielles et parallèles de ceux-ci. Nous introduisons ici une algèbre de Services permettant de définir de manière rigoureuse des telles combinaisons de Services.

Considérons des ensembles de Services {Si} de type (1, 1), de relations $\{R_j\}$, et d'assertions de correspondance <MAP, S_i, R_j, $PRE_{i,j}$, $POST_{i,j}$ > exprimant le fait que la relation $R_j^{PREi,j,\ POSTi,j}$ définit le sens de S_i. Nous définissons les opérations de *Composition*, *Union* et *Restriction* <u>*de Services*</u>, de manière symétrique à celles définies sur les relations.

Définition (composition, union, et restriction de services). *Soit des Services* S_1, S_2 *et S, et deux prédicats PRE et POST respectivement définis sur In (S) et Out (S) par :*

$S_1 * S_2 : o'' \in (S_1 * S_2)(o)$ *ssi il existe o' tel que* $o' \in S_1(o)$ *ET* $o'' \in S_2(o')$

$S_1 + S_2 : o' \in (S_1 + S_2)(o)$ *ssi* $o' \in S_1(o)$ *OU* $o' \in S_2(o)$

$S^{PRE, POST}: o' \in S^{PRE, POST}(o)$ *ssi* $o' \in S(o)$ *ET PRE(o) ET POST(o')*

Nous avons : In ($S_1 * S_2$) = In (S_1), Out ($S_1 * S_2$) = Out (S_2), et In ($S_1 + S_2$) = In (S_1). In (S_2), Out ($S_1 + S_2$) = Out (S_1) + Out (S_2). Il est à noter que l'évaluation de ($S_1 * S_2$)(o) conduit à une invocation de S_1, mais à un nombre d'invocations de S_2 dépendant de la cardinalité du résultat $\{S_2(o)\}$.

---

1. Il existe des cas où cette correspondance partielle n'obéit pas à un tel critère de rationalité, par exemple d'une base de donnée répliquée de manière asynchrone, et dont la mise à jour peut refléter de manière seulement partielle le contenu de référence à un instant donné. Ce cas relève dans notre approche de la notion de QoS.

2. On notera que la notion de *précondition* ici définie n'est pas ici la condition d'activation du Service, mais la condition attendue sur un individu en entrée pour que le Service réalise la Relation. On notera de même que la notion de *postcondition* ici définie n'est pas la notion d'*effet* (les Services ici considérés sont sans effet de bord).

Nous introduisons également les *filtres* $F_{[C1, c2]}$, qui sont des Services prédéfinis *réalisant* les relations de filtrage de restriction de la forme $[C_1, C_2]$ d'une classe sur une autre. Nous pouvons considérer une expression algébrique de Services comme un graphe d'exécution, où les instructions élémentaires sont des invocations de services, orchestrées par les opérateurs de composition, d'union et de restriction, qui représentent respectivement la séquence, l'activation parallèle (« fork and join ») et la condition de test. Les Services fournis par l'infrastructure seront appelés Services *réels*, les Services définis par une expression algébrique de Services *réels* étant appelés Services *abstraits*.

Si <MAP, S_1, R_1> ET <MAP, S_2, R_2> , nous avons <MAP, $S_1 * S_2$, $R_1 * R_2$>, et <MAP, $S_1 + S_2$, $R_1 + R_2$>. Par ailleurs, si <MAP, S, R>, alors on a : <MAP, $S^{PRE, POST}$, $R^{PRE, POST}$>. Ceci définit un *homomorphisme* algébrique entre l'ensemble des relations et celui des Services de type (1, 1). Un résultat important concerne l'évaluation de relations *dérivées* à partir de plusieurs services réalisant des correspondances partielles grâce au résultat suivant.

Soit une relation R, et un ensemble de services S_i *, i =1, …, n, tel que <MAP,* S_i*,* R*,* PRE_i*,* $POST_i$*>. Le Service* $S = \sum_{i = 1, \ldots, n} S_i$ *réalise la relation R, soit <MAP,* $\sum_{i = 1, \ldots, n} S_i$*, R> ssi OU* $_{PREi\ ,\ POSTi}$ *(*PRE_i *ET* $POST_i$*) = True*

Ceci correspond intuitivement au fait que, pour que la relation puisse être évaluée dans tous les cas et de manière exhaustive, toutes les combinaisons des sous-ensembles des entrées et des sorties doivent être réalisées par un Service

5.5. *Correspondances complexes*

Nous considérons maintenant le cas de **Services (n, p)** (n paramètres d'entrée, p paramètres de sorties). Pour de tels Services, le modèle de correspondance précédent n'est plus directement applicable. L'un des apports de notre algèbre est de permettre la définition de relations permettant de capturer la sémantique de tels Services, et autorisant donc leur indexation et leur réutilisation.

5.5.1. *Services à n entrées et une sortie*

Considérons par exemple les classes *Patient*, *DossierMedical* et *Etablissement*, liées par les relations *aDossier* (un ou plusieurs dossiers pour un Patient) et *géréPar* (un seul établissement pour un dossier donné), extrait du cas d'étude (figure 5).

Considérons par ailleurs un Service offert par un Serveur d'Identité Régional délivrant les identifiants des dossiers à partir du couple (Patient, Etablissement), soit S : (Patient, Etablissement) → $2^{DossierMédical}$. S est un Service (2, 1), avec In_1 (S) = *Patient*, In_2 (S) = *Etablissement*, et Out (S) = *DossierMédical*.

On remarque que la sémantique de S est directement donnée par la relation R_S définie ci-après, reliant un patient P à un ensemble de dossiers {D}, où E est un individu (Etablissement) paramétrant la relation.

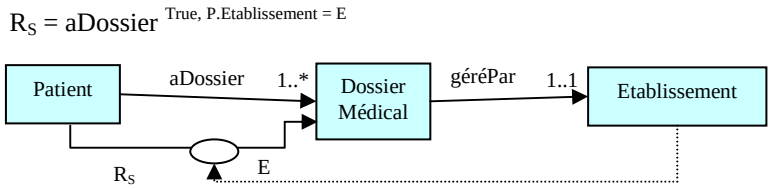


Figure 5. *Illustration d'une correspondance complexe*

En d'autres termes, R_S est simplement une restriction de la relation *aDossier* , restreinte par le biais d'un prédicat POST paramétré par l'autre argument d'entrée E. Ceci peut s'écrire <MAP, $S_{in2 = E,}$ aDossier, True, P.Etablissement = In_2 (S) >.

où $S_{in2 = E}$ est le Service (1, 1) associant à un patient P la sortie S (P, E), et où In_2 (S) désigne la valeur du second argument de S, c'est-à-dire la valeur courante de l'entrée E. De manière générale, la sémantique d'un Service (n , 1), $S : (C_{in\ 1} \ldots C_{in\ n}) \rightarrow 2^{Cout}$, peut être définie grâce une formule algébrique du type :

R_S = EXPR ({R_k}, {PR_l})

définissant la relation *étendue* R_S reliant une sortie out à une entrée d'indice i_0, et exprimée à partir de relations de l'ontologie et de prédicats {PR_l} utilisant les valeurs des arguments d'entreé $C_{in\ 1}$, …., $C_{in\ n}$. R_S est la relation *réalisée* par le Service S. Cette expression n'est pas nécessairement unique, en particulier dans le cas où il existe dans l'ontologie des relations inverses. La démarche générale pour déterminer une telle expression consiste à : 1) déterminer un chemin de relations allant d'une classe présente en argument d'entrée à la classe présente en sortie, et 2) ajouter les contraintes correspondant à la donnée des autres arguments d'entrée.

5.5.2. *Services à n entrées et p sorties*

Le cas général d'un Service (n, p) : $S : (C_{in\ 1}, \ldots., C_{in\ n}) \rightarrow (2^{Cout\ 1}, \ldots, 2^{Cout\ p})$ se déduit immédiatement des résultats précédents. La sémantique de S est en effet parfaitement définie par la sémantique des P Services partiels :

$S_j : (C_{in\ 1}, \ldots., C_{in\ n}) \rightarrow 2^{Cout\ j}$

qui sont les p projections de S sur les p espaces de sorties $C_{out\ 1}$, …, $C_{out\ p}$. ce qui ramène à la définition de la sémantique de Services de type (n, 1).

5.6. *Qualité de service*

Dans le cas général, il peut exister différentes manières de composer des Services pour évaluer une même propriété. Le choix opportun doit se baser *in fine* sur la qualité de service (QoS) globale résultante. La notion de QoS comprend plusieurs dimensions (délai**,** disponibilité**,** accessibilité**,** sécurité**,** etc.) à chacune

desquelles doit être associée une métrique. Nous considérons ainsi un ensemble de facteurs de Qualité $F_{i,}$ i=1, …, p, avec des métriques q_i associées, et une *fonction de qualité* : $q = [q_1, .., q_p]$, associant à tout Service S un vecteur à p dimensions $[q_1, .., q_p]$, où la i-ème dimension est une quantification du facteur $F_{i.}$. Nous considérons également une *fonction de préférence*, $\Lambda (q) = \Lambda ([q_1, .., q_p]) = \sum_{i=1, ..., p} \alpha_i.q_i$, fournie par l'entité appelante, et définissant la manière d'agréger les différents facteurs en une valeur unique. Cette fonction définit le besoin de l'entité appelante et donne une mesure objective permettant de comparer différentes *réalisations* possibles d'une propriété en termes d'invocations de Services

A chaque facteur q_i, on associe des lois permettant d'agréger les facteurs de chaque Service opérande, lorsque les Services invoqués sont combinés par compositions et unions, soit : $q (S_1 * S_2) = F (q(S_1), q(S_2)) = [F_i (q_i(S_1), q_i(S_2))]$, et $q (S_1 + S_2) = G (q(S_1), q(S_2)) = [G_i(q_i (S_1), q_i(S_2))]$

Les fonctions F_i et G_i (somme, max, min, etc.) dépendent de la sémantique attachée au facteur considéré. Dans la cas de la composition, où le nombre d'invocations du Service S_2 dépend de la cardinalité du résultat délivré par S_1, la règle implémentée par la fonction F doit être basée sur une stratégie d'évaluation telle l'espérance mathématique ou le minimax. Pour simplifier les notations, nous écrirons $q (S_1 * S_2) = q (S_1) * q (S_2)$, et $q (S_1 + S_2) = q (S_1) + q (S_2)$, opérations qui expriment la combinaison des vecteurs qualité de service par les opérateurs * et +.

6. Orchestrations de services et génération de plans

Dans le cas général, c'est l'activation de plusieurs Services qui permettra de délivrer l'information recherchée, et il faudra déterminer les Services à invoquer ainsi que le plan d'orchestration optimum vis-à-vis de la QoS. Dans un premier temps, nous définissons les différents problèmes posés. Nous présentons ensuite des principes permettant de déduire de nouvelles *correspondances* de celles données au départ. Enfin, sur cette base, nous présentons un algorithme général de résolution.

6.1. *La génération de plans d'orchestration*

Il s'agit de déterminer un plan associé à l'évaluation d'une propriété, c'est-à-dire la transformation de l'expression algébrique initiale définissant la propriété à évaluer en un plan d'invocations de Services, que l'on appellera *le plan d'orchestration*. Celui-ci peut être représenté par un graphe sans cycle où les nœuds représentent des résultats intermédiaires, et où les arcs représentent des invocations de Services. Un tel graphe permet de représenter des invocations de Services séquentielles ou parallèles. Nous faisons le choix ici de centrer notre exposé sur l'évaluation des propriétés de type relation, l'approche décrite restant sur le principe valide pour des requêtes afférant à l'évaluation d'un attribut.

Une configuration initiale est définie par : 1) une *ontologie* du domaine et un ensemble de propriétés *dérivées*, 2) un *ensemble de Services*, avec QoS associées, et 3) un *ensemble d'assertions de correspondance*, exprimant la sémantique des Services. Nous considérons une *relation R à évaluer*, pour une fonction de préférence associée et un *individu x* donné en entrée. Les problèmes posés sont :

1) déterminer si, dans la configuration, il existe ou non *un plan d'orchestration de Services* qui délivre le résultat attendu pour tout individu x ϵ dom (R) ;

2) dans le cas où il existe une solution générale (c'est-à-dire de solution valable pour tout x), *construire le ou les graphes solutions*, et, s'il y a plusieurs solutions, déterminer la solution optimale au sens de la QoS et de la fonction de préférence ;

3) dans le cas où il n'existe pas de solution générale, déterminer *sous quelles conditions éventuelles* restreignant l'entrée x il sera possible d'obtenir une solution.

Le second problème est celui de la recherche d'une solution *uniformément optimale* sur tout dom (R). Si cette solution existe, elle définira un *plan d'orchestration optimal* qui sera mémorisé comme applicable à tout évaluation future <R, x, ?y> (*plan d'orchestration statique*). Si elle n'existe pas, il pourra cependant exister des solutions optimales pour des sous-classes de type $(\text{dom}(R))^P$. Le plan convenable devra être activé en fonction de conditions sur l'entré x (*plan d'orchestration dynamique*). Nous proposons dans la suite de la section une approche qui permettra de traiter conjointement les différents problèmes par le biais d'un algorithme unique.

Un *graphe d'orchestration* G_i définit une expression algébrique de Services. Un tel graphe pourra réaliser complètement la propriété à évaluer. Ce sera alors une solution. Nous considérerons aussi au cours du déroulement de l'algorithme des graphes d'orchestration qui réaliseront seulement partiellement l'expression algébrique définissant la propriété à évaluer.

Un graphe d'orchestration G_i a une origine Orig (G_i), qui est un ensemble d'individus étiqueté par un prédicat PRE (G_i) déterminant la condition qui doit être vérifiée sur les entrées pour que le plan soit valide. Il a également une terminaison End (G_i) qui est une ensemble d'individus, étiqueté par un prédicat POST (G_i), qui détermine une possible limitation des résultats attendus. A un graphe d'orchestration, est associée également une valeur de QoS Q(G_i).

Un graphe G_i avec la classe d'entrée dom (R) comme origine, la classe de sortie range (R) comme terminaison, et vérifiant POST (G_i) = True, sera appelé une *solution partielle candidate*. Si, de plus, PRE = True, alors G_i sera une *solution candidate*. Si nous souhaitons déterminer par avance un plan d'orchestration d'une propriété (équivalent d'une requête compilée), nous appliquons l'algorithme. S'il existe une ou plusieurs solutions candidates, alors la solution au problème est la solution candidate G_0 qui optimise QoS (G). Cette solution G_0 permettra d'évaluer la propriété cible pour n'importe quel individu d'entrée.

S'il n'existe pas de *solution candidate*, mais s'il existe seulement des *solutions partielles candidates* $\{G_j\}$, alors il sera possible d'évaluer la propriété cible à l'aide d'un G_j particulier ssi l'individu donné en entrée x vérifie bien la contrainte PRE (G_j). Les solutions fournies par l'algorithme seront alors les différentes *solutions partielles candidates* G_1 optimisant QoS (G_j) sur l'espace des solutions partielles candidates telles que PRE (G_j) (x) soit vérifié. A la différence du cas précédent, il y a seulement un *ordre partiel* sur les solutions, faute d'un ordre sur leur domaine de validité. Cet ordre devient total dès que l'individu x en entrée est fixé. Les solutions délivrées par l'algorithme sont les *optima de Pareto* associés à la QoS. Pour un x_0 donné, soit il n'y aura pas de solution, soit il existera une solution optimale, mais dont l'optimalité n'est garantie que pour le seul x_0.

6.2. *Propriétés*

Le principe de notre algorithme est celui d'une génération itérative de nouvelles assertions de *correspondances* dérivées de celles déjà identifiées. Le problème est d'identifier les cas dans lesquels il est possible de générer de telles nouvelles *correspondances*. Nous présentons ici des propriétés qui le permettent. Soit deux relations R_1 et R_2, et deux Services S_1 et S_2, tels que <MAP, S_1, R_1, PRE_1, $POST_1$> et <MAP, S_2, R_2, PRE_2, $POST_2$>. Pour caractériser $S_1 * S_2$ et $S_1 + S_2$ en termes de correspondances, nous utilisons les propriétés suivantes.

Propriété 1. On peut définir une assertion de correspondance pour $S_1 * S_2$ ssi $POST_2$ = True AND PRE_1 = True, et cette correspondance est : <MAP, $S_{1*} S_2$, $R_{1*} R_2$, PRE_1, $POST_2$>. Comme cas particulier, nous pouvons remarquer que, si S_1 réalise R_1 ($PRE_1 = POST_1$ = True) et si S_2 réalise R_2 ($PRE_2 = POST_2$ = True), alors $S_{1*} S_2$ réalise $R_{1*} R_2$

Propriété 2. la meilleure correspondance qui puisse être définie pour $S_1 + S_2$ est : <MAP, ($S_1 + S_2$), PRE_1 OR PRE_2, $POST_1$ AND $POST_2$>. Ainsi, la correspondance existe ssi $POST_1$ ET $POST_2$ n'est pas le prédicat False, c'est-à-dire ssi output (S_1) ∩ output (S_2) ≠ Ø

Exemple. Soit une classe C_1 ayant un attribut A, une classe C_2 ayant un attribut B, une Relation R_1 de C_1 vers C_2, et une Relation R_2 de C_2 vers C_3. Si nous avons des Services $S_{1,1}$, $S_{1,2}$, $S_{2,1}$, $S_{2,2}$, tels que <MAP, $S_{1,1}$, R_1, $PRE_{1,1}$ = "$A=a_0$", True> ; <MAP, $S_{1,2}$, R_1, $PRE_{1,2}$ = "$A\neq a_0$", True> ; <MAP, $S_{2,1}$, R_2, $PRE_{2,1}$ = "$B=b_0$", True> et <MAP, $S_{2,2}$, R_2, $PRE_{2,2}$ = "$B=b_0$", True>. En premier lieu, nous, pouvons générer deux correspondances. La première est <MAP, $S_{1,1} + S_{1,2}$, R_1 , True, True>, c'est-à-dire <MAP, $S_{1,1} + S_{1,2}$, R_1>. La seconde est <MAP, $S_{2,1} + S_{2,2}$, R_2 , True, True>, c'est-à-dire <MAP, $S_{2,1} + S_{2,2}$, R_2 >. Considérant le nouveau Service *abstrait* $S = S_{1,1} + S_{1,2}$, et $S' = S_{2,1} + S_{2,2}$, il apparaît que nous pouvons à nouveau générer une nouvelle correspondance <MAP, $S_{,*}S_{\cdot}$, R_1 , True, True>, c'est-à-dire <MAP, $S_{,*}S_{\cdot}$, R_1>. Au contraire, si nous avons des correspondances telles que <MAP, $S_{1,1}$, R_1, $PRE_{1,1}$ = "$A=a_0$", $POST_{1,1}$ = "$B=b_0$"> et <MAP, $S_{1,2}$, R_1, $PRE_{1,2}$ = "$A\neq a_0$", $POST_{1,2}$ = "$B\neq b_0$">, la seule nouvelle correspondance déductible est <MAP, $S_{2,1} + S_{2,2}$, R_2 >, et aucune autre correspondance ne peut être générée.

Nous définissons de plus comme suit l'équivalence de deux Services :

Définition (équivalence de services). *Soit deux Services S_1 et S_2 et deux assertions de correspondance <MAP, S_i, R_i, PRE_i, $POST_i$>, i = 1, 2. S_1 et S_2 sont dits équivalents ssi $R_1 = R_2$, $PRE_1 = PRE_2$, $POST_1 = POST_2$.*

Nous dirons que $S_1 \geq S_2$ ssi $R_1 = R_2$ ET $PRE_1 \rightarrow PRE_2$ ET $POST_1 \rightarrow POST_2$ (c'est-à-dire que S_1 est une meilleure réalisation que S_2 de la même relation $R = R_1 = R_2$)[3]. Dans une expression algébrique de Services, il sera possible de remplacer un opérande S_i par un Service équivalent S_j, ou encore par un Service S_k tel que $S_i \geq S_k$ si $q(S_k) \geq q(S_i)$.

Pour *réaliser* une expression algébrique de propriétés, il est nécessaire en premier lieu de rechercher dans l'infrastructure tous les Services qui *réalisent* (totalement ou partiellement) une partie de l'expression. Les Services ainsi trouvés par cette opération de *matching*, sont les Services candidats à former les briques de base du futur plan d'orchestration associé à l'évaluation de la relation R.

Le *matching* de Services est enfin défini de la manière suivante.

Définition (matching de service). *Soit une relation R définie par une expression EXPR, et soit un Service S. S matche R ssi il existe une correspondance telle que <MAP, S, R', PRE, POST> , où R' est une sous expression de R.*

6.3. *Un algorithme pour la génération de plans d'orchestration*

Nous présentons ici de manière succincte l'algorithme de génération des graphes d'orchestration pour l'évaluation d'une relation *étendue*, définie par une expression. Selon le cas considéré, l'algorithme fournira 1) un plan d'orchestration *uniformément optimal*, applicable à tout individu x de dom (R) donné en entrée, et, si ce plan optimum n'existe pas, 2) un ensemble de plans avec des contraintes associées sur l'entrée et valeurs des QoS associées à chaque plan. L'algorithme suit plusieurs étapes.

Etape 1) Evaluation de l'expression algébrique en termes de propriétés *natives*. On remplace de manière récursive chaque propriété *dérivée* opérande par sa définition.

Etape 2) Simplification de l'expression algébrique. On réduit le nombre d'opérandes grâce aux propriétés des opérateurs (factorisation). Ceci a pour but de minimiser le nombre d'invocations de Services et le flot de résultats intermédiaires. La formule est mémorisée sous la forme d'un arbre d'exécution, où les feuilles sont les opérandes R_i et les nœuds des résultats partiels.

---

3. La relation de comparaison « ≥ » n'est *pas* une relation d'ordre car $S_1 \geq S_2$ et $S_2 \geq S_1$ implique S_1 équivalent à S_2, mais pas nécessairement $S_1 = S_2$.

Etape 3) Construction du graphe de flot associé à l'expression. Partant de l'expression précédente, on génère un graphe orienté correspondant à l'évaluation progressive du résultat. Dans ce graphe, les noeuds $\{C_i\}$ représentent des collections d'individus correspondant aux différents résultats intermédiaires, et les arcs $\{R_j\}$ sont des instances de relations reliant une collection à une autre, et étiquetés par la relation opérande concernée.

Ce graphe est sans cycle. Sa source est la classe en entrée dom (R) et le puit unique est la collection d'individus résultat (il est à noter qu'une même relation pourra être présente plusieurs fois dans le graphe sous la forme de plusieurs arcs). Par exemple, l'expression $R = R_1 * (R_2 + (R_3 * R_4))$, génère le graphe de flot suivant, qui a 4 nœuds et 4 arcs.

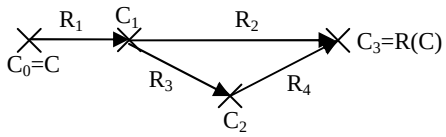


Figure 6. *Exemple de graphe d'exécution*

Etape 4) Collecte des Services applicables à des sous-parties du graphe (*matching*). Cette étape consiste à extraire de la collection de Services existants les Services qui peuvent être en correspondance totale ou partielle avec une partie de l'expression, comme exposé précédemment. On obtient un sous-ensemble $\{S_i\}$ de Services qui est l'ensemble de départ de l'étape finale de l'algorithme.

Etape 5) Génération des graphes candidats. Ceci est fait en combinant de manière itérative les Services de départ, de manière à obtenir, s'il existe, le résultat attendu qui réalise la relation R.

Deux approches sont envisageables pour développer un tel algorithme : 1) une approche descendante, qui part de la relation de départ, et tente de l'exprimer comme une association des Services donnés en entrée. 2) une approche ascendante, qui part des Services donnés en entrée, et les combine itérativement de sorte à dériver à chaque étape de nouvelles correspondances, réalisant une correspondance meilleure et/ou plus complète que celles déjà exhibées. Dans ce second cas, l'algorithme stoppe lorsqu'il n'y a plus ni nouveau *matching* de Service possible, ni possibilité de déduire de meilleures correspondances (au sens de la relation $\geq$) que celles déjà présentes. Cet arrêt est garanti du fait de la croissance stricte d'une fonction sur un ensemble fini. Lorsqu'il existe une correspondance impliquant à la fois le nœud origine et le nœud terminaison du graphe, alors il existe des orchestrations qui sont au moins des *solutions partielles candidates*.

Nous avons adopté cette seconde approche car, si une solution uniformément optimale n'existe pas, cet algorithme permet d'exhiber en sortie les meilleures

solutions *partielles*, ce qui permet de résoudre en un seul algorithme les différents problèmes, soit 1) la détermination de l'existence d'une solution uniformément optimale sur dom (R), 2) la construction de celle-ci lorsqu'elle existe, 3) la détermination des meilleures solutions *partielles* qui permettront des évaluations dynamiques « à la volée » (optimales sur un domaine restreint) de la relation.

Le principe de l'étape 5 est donc le suivant : nous considérons un graphe orienté sans cycle, basé sur le graphe de flot précédent, et dont les nœuds sont ceux du graphe initial, mais avec des arcs additionnels possibles. Il y a ainsi deux types d'arcs : les *arcs initiaux*, représentant les relations opérandes (*arcs relation*), et de *nouveaux arcs* générés itérativement, qui représentent des Services *réels* ou *abstraits réalisant* de manière partielle ou totale une partie de la relation (*arcs Service*). Un tel arc (C_i, C_j) représente un Service qui *réalise* la relation définie par le passage de C_i à C_j. Il est étiqueté par un triplet (PRE, POST, q), où PRE et POST sont des prédicats exprimant le caractère *partiel* possible de la correspondance, et où q est le vecteur de QoS associé au Service. L'algorithme correspondant à cette étape peut être exprimé comme une procédure récursive synthétisée ci-après :

Algorithme
  POUR CHAQUE S DANS Input Service set
    Exprimer le mapping M_i = (R, PRE, POST) entre S et R
    Ajouter un arc Service, étiqueté avec le mapping M_i et le vecteur de QoS de S
    INTEGRER (S)
  FIN POUR CHAQUE
FIN Algorithme
Procedure INTEGRER (in S : Service)
  SI Il existe S' tel que (S' ≥ S) OR ((S' Eq S) AND q (S') ≥ q (S)
  ALORS détruire S
  SINON Associer S avec un arc Service possible E_i
    POUR CHAQUE E_i
      Déterminer le mapping associé (R_i, PRE_i, $POST_i$)
      Déterminer la QoS résultante q_i
      Ajouter un arc Service étiqueté avec R, PRE, POST et q
      INTEGRER (E_i)
    FIN POUR CHAQUE
  FIN SI
FIN Procedure

Les arcs Service redondants (correspondant à des Services *inférieurs* par la relation d'ordre à un service déjà présent, or encore *équivalents* mais avec une QoS plus faible) sont retirés du graphe. Si des solutions partielles existent, alors l'algorithme converge vers la meilleure solution partielle. Ces solutions partielles sont les *arcs Service* (s'ils existent) reliant en fin d'algorithme l'origine du graphe à sa sortie, et étiquetés avec une condition POST = True.

Si l'un de ces arcs est labellisé avec une condition PRE = True, alors il s'agit de la *solution uniformément optimale*. Dans le cas contraire, le résultat de l'algorithme est un ensemble des triplets (S_i, PRE_i, q_i), où S_i est une solution partielle, PRE_i la condition PRE correspondante restreignant l'espace de validité de la solution, et q_i la qualité de service attaché à la solution. Une solution est une expression algébrique des Services d'entrée. La solution à un problème impliquant l'évaluation d'un attribut est une extension simple de l'algorithme précédent.

7. Le prototype

Nous avons développé un prototype validant l'approche sur un cas d'étude considéré. Ce prototype supporte la définition et la gestion de propriétés *dérivées*, définies sur une ontologie OWL de domaine, et permet la génération de plans d'orchestration sur la base de l'algorithme proposé, ainsi que la mémorisation et l'exécution des plans.

Le framework développé comprend: 1) le *registre de web services***,** qui gère les descriptions sémantiques des Services avec QoS associées, par le biais d'annotations complétant les descriptions WSDL ; 2) le *gestionnaire de propriétés dérivées*, gérant les définitions des relations et attributs dérivés (formules algébriques définies dans un langage inspiré de MathML) ; et 3) la *plate-forme de web service*, qui permet de développer, enregistrer, et exécuter des Web Services. A ceci, s'ajoutent deux composants logiciels : le *Générateur*, qui établit des plans d'orchestration sur la base de l'algorithme proposé, mémorisés sous la forme de descriptions BPEL, et l'*Orchestrateur*, qui pilote l'exécution des plans générés.

L'ontologie de domaine considérée repose sur différents standards (HL7, CIM10 pour les pathologies, CCAM pour les actes). A l'exécution, une application cliente accède à la bibliothèque de propriétés dérivées et active si nécessaire le Générateur qui détermine alors le plan optimal relatif à la propriété choisie. L'application invoque alors l'orchestrateur en lui communiquant une référence à ce plan (requête dynamique), ou à un plan précédemment déterminé (requête statique).

Sur cette base, nous avons mis en œuvre l'approche dans le contexte de la fédération de systèmes d'information de santé, sur la base de données anonymisées. Plus précisément, trois cas représentatifs du domaine de l'oncologie ont été traités : 1) l'accès à des dossiers patient complets à partir d'un identifiant de patient, destiné aux unités de concertation pluridisciplinaires, 2) l'accès à des synthèses épidémiologiques récapitulant les cas rencontrés pour une pathologie et une période données, et 3) l'accès à des cas médicaux anonymes à l'attention des étudiants.

8. Conclusion

L'intégration de sources d'information hétérogènes et leur interrogation par le biais d'orchestrations de web services, ne sont pas des problèmes récents. Mais avec l'importance croissante des ontologies et des technologies du web sémantique, ces problèmes constituent aujourd'hui un nouveau défi.

Dans cet article, considérant le fait que la sémantique d'un Service d'accès à des informations peut être valablement définie par le biais de concepts d'une ontologie du domaine, en particulier des relations, nous montrons que, sous cette hypothèse, il est possible de construire un modèle consistant et bien formalisé, permettant de supporter les définitions de nouvelles propriétés, combinaisons et évaluations effectives de celle-ci comme des opérations algébriques. Nous montrons également que ce formalisme permet de construire des plans d'invocations de Services dotés de propriétés intéressantes d'optimalité vis-à-vis de la qualité de service.

Les résultats sont encourageants dans la mesure où tous les principes proposés apportent une importante capacité d'ouverture et de flexibilité. L'expérimentation montre que l'approche constitue un moyen efficace, dans le cadre des hypothèses adoptées, pour 1) intégrer facilement de nouvelles entités participantes, et 2) prendre en compte un nouveau besoin en information sans développements spécifiques.

Sur la base des résultats obtenus, plusieurs axes de recherche s'ouvrent à ce jour. Un premier axe consiste à étendre l'approche à des formes plus générales de propriétés, en particulier à celles *déduites* par le biais de règles. Ceci revient à introduire une composante déductive dans le modèle. Un second axe consiste à étendre l'approche en considérant, non plus seulement les services d'accès, mais également les Services ayant un *effet* sur les données (création, modification) ou sur le monde réel. Enfin, une voie de poursuite consiste à inscrire cette approche dans le cadre des standards et langages actuellement proposés (OWL, OWL-S) et être ainsi en mesure de proposer des langages déclaratifs adaptés.

9. Bibliographie

Andrews T., Curbera F., Dholakia H., Goland Y., Klein J., Leymann F., Liu K., Roller D., Smith D., Thatte S., Trickovic I., Weerawarana S., "Business process execution language for web services (bpel4ws)", version 1.1, mai 2003.

Akkiraju R., Farrell J., Miller J., Nagarajan M., Schmidt M. T., Sheth A., Verma K., Web service semantics-wsdl-s, a joint uga-ibm technical note, version 1.0, Technical report, April 2005.

Al-Masri E. and Mahmoud Q., "QoS-based Discovery and Ranking of Web Services", *Proceedings of the 16th International Conference on Computer Communications and Networks*, IEEE ICCCN 2007, Honolulu, Hawaii, USA, 13-16 août 2007, p. 529-534.

Banerji A., Bartolini C., Beringer D., Chopella V., Govindarajan K., Karp A., Kuno H., Lemon M., Pogossiants G., Sharma S., Williams S., "Web services conversation language (wscl) 1.0", *W3c note, World Wide Web Consortium*, mars 2002.

Casati F., Ilnicki S., Jin L. J., Krishnamoorthy V., Shan M. C., "Adaptive and dynamic service composition in eflow", *Proc. of the 12th International Conference on Advanced Information Systems Engineering (CAiSE'00)*, 2000, Londres, UK, Springer-Verlag, p. 13-31

Dong X., Halevy A., Madhavan J., Nemes E., Zhang J., "Similarity search for web services", *Proc. of Very Large Data Bases (VLDB'04)*, 2004, p. 372-383.

Fensel D., Bussler C., "The web service modelling framework wsmf", *Electronic Commerce Research and Applications*, 2002, p. 113-137.

Grimm S., *Discovery Identifying relevant services*, Springer, 2007, p. 211-244.

Haller A., Cimpian E., Mocan A., Oren E., Bussler C., "Wsmx - a semantic service-oriented architecture", *Proc. of the International Conference on Web Service (ICWS 2005)*, 2005, p. 321-328.

Klusch M., Fries B., Sycara K., "Automated semantic web service discovery with owls-mx", *Proc. of the international joint conference on Autonomous agents and multiagent systems (AAMAS '06)*, New York, USA, 2006, ACM. p. 915-922.

Klusch M., Kapahnke P., "Semantic web service selection with sawsdl-mx", *Proc. of the Second International Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web*, Karlsruhe, Allemagne, 27 octobre 2008.

Klusch M. and FKaufer F., "Wsmo-mx : A hybrid semantic web service matchmaker", *Web Intelligence and Agent Systems*, vol. 7, n° 1, 2009, p. 23-42.

Kopeck J., Vitvar T., Bournez C., Farrell J., "Sawsdl : Semantic annotations for wsdl and xml schema", *IEEE Internet Computing*, vol. 11, n° 6, 2007, p. 60-67.

Laukkanen M. Helin H., "Composing workows of semantic web services", *Proc. of the Workshop on Web Services and Agent-based Engineering*, 2003.

Meditskos G., Bassiliades N., "Structural and role-oriented web service discovery with taxonomies in owl-s", *IEEE Trans. on Knowl. and Data Eng.*, vol. 22, n°2, 2010, p. 278-290.

Martin D., Burstein M., Mcdermott D., Mcilraith S., Paolucci M., Sycara K., Mcguinness D., Sirin E., Srinivasan N., "Bringing semantics to web services with owl-s", *World Wide Web*, vol. 10, n° 3, 2007, p. 243-277.

McIlraith S., Son T., and Zeng H., "Semantic web services", IE*EE Intelligent Systems. Special Issue on the Semantic Web*, vol. 16, n°2, mars-avril 2001, p. 46-53.

Osman T., Thakker D. and Al-Dabass D., "Bridging the Gap between Workflow and Semantic-basedWeb services Composition", *Proc of WWW Service composition with Smentic Web Services (wscomp05)*, Compiègne, France, 19 septembre 2005, p. 13-23.

Papazoglou M., "Service-oriented computing: Concepts, characteristics and directions", *Proc. of the Fourth International Conference on Web Information Systems Engineering (WIDE'03)*, Washington, DC, USA, 2003. IEEE Computer Society, p. 3.

Paolucci M., Kawamura T., Payne T., and Sycara K., "Semantic matching of web services capabilities", *Proc. of the International Semantic Web Conference (ISWC)*, 2002, p. 333-347.

Patil A., Oundhakar S., Sheth A., and Verma K., "Meteor-s web service annotation framework", *Proc. of the 13th international conference on World Wide Web (WWW'04)*, New York, USA, 2004, ACM, p. 553-562.

Ran S., "A model for web services discovery with qos", *SIGecom Exch.*, vol. 4, n° 1, 2003, p. 1-10.

Roman D., Keller U., Lausen H., de Bruijn J., Lara R., Stollberg M., Polleres, Feier C., Bussler C., and Fensel D., "Web service modeling ontology", *Applied Ontology*, vol. 1, n° 1, 2005, p.77-106.

Stollberg M., Keller U., Lausen H., and Heymans S., "Two-phase web service discovery based on rich functional descriptions", *Proc. of the 4th European Semantic Web Conference (ESWC 2007)*, Springer, 2007.

Srinivasan N., Paolucci M., and Sycara K., "An efficient algorithm for owl-s based semantic search in uddi", *Proc. of the First International Workshop Semantic Web Services and Web Process Composition*, 2004, p. 96-110.