Un modèle de composition des services web sémantiques

N. TEMGLIT[1], H.ALIANE[2], M.AHMED NACER[3]

Division Théorie et Ingénierie des systèmes Informatiques(DTISI)
Centre de Recherche sur l'Information Scientifique et Technique (CERIST)
ALGERIE

[1]t_nacera@yahoo.fr, [2] haliane@cerist.dz

[3]Laboratoire des Systèmes Informatiques(LSI)
Université des Sciences et de la Technologie Houari Boumediene (USTHB)
ALGERIE

[3]anacer@cerist.dz

RÉSUMÉ. Le travail présenté ici vise à proposer un modèle pour la composition des services web sémantiques. Ce modèle est basé sur une représentation sémantique de l'ensemble des concepts manipulés par les services web d'un domaine d'application, à savoir, les opérations et les concepts statiques utilisés pour décrire les propriétés des services web. Différents niveaux d'abstraction sont donnés au concept opération pour permettre un accès progressif aux services concrets. Ainsi, deux plans de composition à granularités différentes (abstrait et concrets) sont générés. Ceci permettra de réutiliser des plans déjà construits pour répondre à des besoins similaires et même avec des préférences modifiées.

ABSTRACT. The work presented here aims to provide a composition model of semantic web services. This model is based on a semantic representation of domain concepts handled by web services, namely, operations and the static concepts used to describe static properties of Web services. Different levels of abstraction are given to the concept of operation to allow gradual access to concret services. Thus, two different levels of the composition plan are generated (abstract and concret). This will reuse plans already constructed to meet similar needs even with modified preferences.

MOTS-CLÉS : composition de services web, matching sémantique, template de composition.

KEYWORDS: Web Service composition, semantic matching, composition template.

1. Introduction

L'évolution d'Internet et la compétitivité entre les entreprises ont été les facteurs de l'explosion des services Web. En effet, les services web peuvent constituer un apport de rapidité et d'efficacité pour l'e-business. La notion de service web désigne essentiellement une application (un programme) mise à disposition sur Internet par un fournisseur de service, et accessible par des clients à travers des protocoles Internet standards. Leurs particularités par rapport aux autres technologies de l'informatique répartie résident dans le fait qu'ils offrent un modèle de composants à couplage faible en utilisant la technologie Internet comme infrastructure pour la communication. La composition des services web constitue une évolution naturelle de cette technologie qui permet de mettre en place des composants services Web au profit de l'intégration d'applications sur le web afin d'atteindre de meilleures solutions.

2. Le modèle de base des Services web

Le modèle des services Web repose sur une architecture orientée service. Celle-ci fait intervenir trois catégories d'acteurs : les fournisseurs de services (i.e. les entités responsables du service Web), les clients qui servent d'intermédiaires aux utilisateurs de services et les annuaires qui offrent aux fournisseurs la capacité de publier leurs services et aux clients le moyen de localiser leurs besoins en terme de services. La dynamique entre ces trois acteurs inclut donc les opérations de *publication*, de *recherche* et de *liens* (binding) d'opérations. Cette dynamique est normalisée à travers 3 standards : un protocole abstrait de description et de structuration des messages, SOAP **[16]**, une spécification XML qui permet la publication et localisation des services dans les annuaires, UDDI **[17]** et un format de description des services Web, WSDL **[20]**. Un service WSDL est composé d'un ensemble d'opérations élémentaires, chacune décrite par un flux de messages échangés entre le client et le service **[7].**

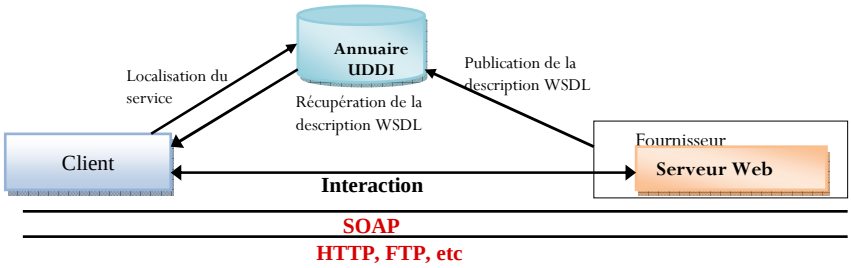

Figure 1. *Modèle de fonctionnement de l'architecture Services Web*

Les services Web constituent un cadre robuste pour assurer l'interopérabilité entre des applications hétérogènes, accessibles en ligne, car ils proposent une représentation homogène du comportement observable du service (i.e. du point de vue du client).

3. L'ajout de la couche sémantique

L'infrastructure de base autour des standards SOAP, WSDL, UDDI est suffisante pour mettre en place des composants interopérables et intégrables mais insuffisante pour rendre automatique et efficace plusieurs tâches liées au cycle de vie des services web, par exemple : la composition et aussi la découverte des services requis. En particulier, WSDL fournit une description concrète mais de bas niveau d'un service Web, en termes de sa localisation, des opérations disponibles et des messages associés ainsi que des types de données et formats de leurs paramètres d'E/S. Ces descriptions sont insuffisantes pour qu'un agent logiciel puisse interpréter la signification réelle des opérations WSDL. Par conséquent, les technologies du web sémantique telles que les ontologies peuvent jouer un rôle prépondérant pour permettre d'expliciter la sémantique des services en facilitant ainsi leur utilisation automatique.
Citons une des principales approches qui avaient conduit le développement des cadres sémantiques des services Web, l'approche OWL-S **[10]** (Ontology Web Language for Web Service) qui est sur le point de standardisation par le W3C. Elle se base sur l'utilisation d'un langage ontologique appelé OWL-S. OWL-S respecte une syntaxe XML et une sémantique de OWL **[9]** (DAML-OIL respectivement) qui est lui même inspiré de la logique de description. A travers trois classes d'informations : ServiceProfile, ServiceModel et le ServiceGrounding, OWL-S permet de fournir à l'agent logiciel deux types de connaissances concernant un service web:

a) Qu'est ce que le service a besoin de l'agent et qu'est ce qu'il peut lui offrir ? la classe ServicePROFILE permet de décrire les capacités et les paramètres à fournir au service pour qu'il puisse travailler.
b) Comment il travaille ? La classe ServiceMODEL permet de décrire le Workflow et les chemins d'exécution possibles d'un service web.

4. Problématique de la composition des services web

La composition des services web se réfère au processus de création d'un service composite offrant une nouvelle fonctionnalité, à partir de services web existants plus simples, par le processus de découverte dynamique, d'intégration et d'exécution de ces services dans un ordre bien défini afin de satisfaire un besoin bien déterminé **[6].** A titre d'exemple, nous pouvons modéliser le service de planification de voyage d'une

agence de voyages comme la composition de plusieurs Services web atomiques (basiques) appartenant à des organisations différentes : réservation de vol, réservation d'hôtel et allocation de véhicule.

Un système de composition comprend principalement trois types de participants (acteurs): Le fournisseur, le consommateur et le composeur. Généralement, le composeur se divise en d'autres sous rôles qui peuvent être les suivants : *traducteur*, *générateur de processus* et l'é*valuateur* **[15]**. Le traducteur traduit la description entre un langage externe utilisé par les participants et un langage interne utilisé par le générateur de processus. Ce dernier, pour chaque requête essaye de générer un plan combinant les services disponibles dans le dépôt de services qui pourront répondre au mieux aux besoins de la requête. Du moment que différents services web peuvent implémenter des fonctionnalités similaires, plusieurs plans de composition peuvent être ainsi générés. Ces plans seront soumis à l'évaluateur qui va sélectionner le meilleur plan pour l'exécution en se basant sur les préférences du demandeur. Enfin, le moteur d'exécution, exécute le plan ainsi sélectionné et retourne le résultat au service demandeur. L'exécution d'un service composite peut être vue comme l'échange d'une séquence de messages entre les services participants selon le modèle de processus défini.

Malgré les efforts de recherche et de développement autour de la problématique de la composition des services, elle reste une tâche hautement complexe et pose un certain nombre de défis. Sa complexité provient généralement des sources suivantes :

- L'augmentation dramatique du nombre des services web sur le web rend très difficile la recherche et la sélection des services web pouvant répondre à un besoin donné.
- Les services sont crées et mis à jour de façon hautement dynamique.
- Les services web sont d'habitude développés par différentes organisations qui utilisent différents modèles conceptuels pour décrire les caractéristiques des services web **[15]**.
- La modularité constitue une caractéristique importante des services Web, par conséquent, les services Web composites doivent garder récursivement les mêmes caractéristiques que les services Web basiques à savoir auto-descriptifs, interopérables, et facilement intégrables **[7]**.

5. Travaux attachés

Selon les travaux effectués dans le champ des services web, on peut classifier la composition selon différents points de vue : manuelle, semi-automatique et automatique ou bien statique et dynamique. Dans le cas de la composition statique, le demandeur doit définir a priori le modèle abstrait de processus décrivant le plan de

composition, par exemple, sous forme d'un graphe de tâches. Dans ce type d'approche, uniquement la sélection et la liaison aux services basiques se font de manière automatique, citons en l'occurrence le système **eFlow [3]**. Par contre, la composition dynamique vise à créer le modèle de processus abstrait, sélectionner et lier les services atomiques aux tâches abstraites de manière automatique et à la demande **[15].** Ce type de composition soulève un défi pour la communauté web sémantique qui essaye d'appliquer des techniques de planification de l'IA pour générer le plan de composition de façon automatique. Nous citons, en l'occurrence, les travaux suivants **[5] [8] [4] [13] [6] [18] [19] [14]**.

Les approches proposées par les différentes communautés possèdent différents niveaux d'automatisation du processus de composition. Cependant, nous ne pouvons pas affirmer que celle qui possède un très haut degré d'automatisation est la meilleure, car, l'environnement des services web est hautement complexe et il n'est pas toujours possible de tout générer de façon automatique et efficace.

D'autres problèmes concernant différents aspects de la composition peuvent être soulevés. En particulier, les méthodes citées auparavant ne prennent pas suffisamment en compte le problème d'interaction avec un utilisateur voulant exécuter un service composite, sachant que les services web concernés sont a priori inconnus et dispersés à l'intérieur d'un vaste entrepôt de services.

6. Un Modèle de composition des services web

Le but de notre travail est de définir une approche progressive de composition des services web sémantiques en partant d'une requête déclarative simple spécifiée par un utilisateur non-spécialiste. Ce dernier aura simplement à spécifier son besoin en terme de tâches à effectuer, il ne sera pas contraint d'être conscient de tous les détails techniques tels que les services concrets participants et la manière dont ils seront composés. La découverte des services participants, leur composition ainsi que leur interopération doivent être effectuées de façon automatique et transparente vis-à-vis de l'utilisateur. Pour ce faire, nous avons basé notre modèle sur :

- Un support sémantique comprenant la description des concepts dynamiques (opérations manipulées par les services web) et les concepts statiques utilisés pour décrire les propriétés des services web.
- Un schéma de mise en correspondance qui permettra de trouver, de manière progressive, une correspondance entre la requête de l'utilisateur et l'ensemble des services publiés.

Ajoutant à cela, le modèle de composition proposé permettra aussi de générer des plans à deux niveaux de granularités (abstrait et concret) et cela pour répondre aux éventuels

problèmes de disponibilité des services et permettre aussi la réutilisation des plans déjà construits pour répondre à des demandes similaires même si certaines préférences ont été modifiées.

6.1. Le support sémantique

L'organisation et la description sémantique de l'espace des services web est un besoin crucial pour permettre une interaction potentielle entre l'utilisateur et l'ensemble des services web disponibles. Les services web tel qu'ils sont définis et présentés sur le web ne permettent pas à leurs utilisateurs de découvrir les fonctionnalités complexes qu'ils peuvent offrir. Pour cela, nous avons basé notre modèle sur une description ontologique qui distingue, d'un coté, différents niveaux d'abstraction des opérations du domaine d'application. D'un autre coté, elle permet de décrire, de manière hiérarchique, les concepts statiques des services web.

6.1.1 Les opérations : Nous représentons les opérations du domaine selon 3 abstractions différentes : Opération abstraite générique, opération abstraite simple et opération concrète. Cette distinction a pour but d'élever le niveau d'abstraction de telle manière que l'utilisateur pourra spécifier sa requête en termes de tâches à accomplir et non pas en termes de services qu'il doit invoquer (figure 3.).

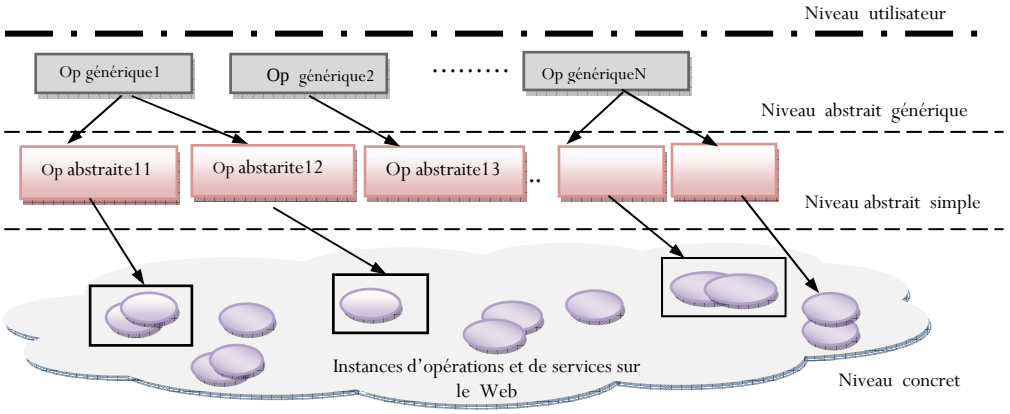

Figure 3. *Les trois niveaux d'abstraction des opérations du domaine*

- Opération générique est une opération ou tâche abstraite qui peut être décomposée en opérations plus simples. Les opérations génériques représentent les traitements métiers les plus fréquemment sollicités dans un domaine d'application donné, on peut dire que ce sont des processus métiers connus qui répondent à des besoins communs des utilisateurs.

Une opération générique est décrite donc par les attributs : Catégorie [domaine, synonymes], Fonction [nom-fonctionnalité, synonymes], E/Sorties [nom, synonymes, type] et une Description textuelle de l'opération. Par exemple, l'opération de réservation de vol peut être décomposée en plusieurs sous opérations :

Catégorie: *[flight-opération]*, Fonction : *[flight-booking, flight-reservation]*,
Inputs: *[departure-date], [return-date], [from, departure-city], [to, arrival-city]*,
Outputs: *[airline], [flight-Nr], [Seat-Nr]*.

- Opération abstraite est une opération qui décrit de façon abstraite une fonctionnalité qui peut être implémentée par plusieurs opérations concrètes appartenant à des services différents. Elle possède une description similaire à celle d'une opération générique à savoir la *catégorie*, la *fonction*, une *description* et *les paramètres d'Entrées/Sortie.* Contrairement à une opération générique, une opération abstraite n'est pas décomposable en d'autres sous-opérations mais elle est directement instanciée en opération(s) concrète(s). Par exemple : la combinaison des deux opérations décrites ci-dessous produit l'opération générique citée auparavant (flight-reservation):

Première opération: Catégorie : *[flight-opération]*, Fonction : *[flight-lookup, serachFor-flight]*, Inputs: *[departure-date], [arrival-date], [from, origin, departure-city], [to, destination, arrival-city]*, Outputs: *[airline], [flight-Nr]*.

Deuxième opération: Catégorie : *[flight-opération]*, Fonction: *[buy-flightTicket]*, Inputs: *[price], [credit-card-Nb], [flight-Nb]*, Outputs: *[buyTicket-confirmation], [..]*.

- Opération concrète est une opération appartenant à un service web concret et disponible sur le web, donc elle possède une implémentation réelle de sa fonctionnalité. Sa description hérite celle de l'opération abstraite avec quelques extensions décrivant les détails d'implémentation et d'autres informations décrivant sa qualité. D'un coté, les détails d'implémentation concernent les informations de *Binding (liaison)* qui précisent le format des messages et les protocoles réseaux utilisés pour l'invocation effective de l'opération (tels que SOAP/HTTP ou SOAP/MIME). D'un autre coté, la qualité d'une opération recouvre trois aspects définis selon différentes métriques: la qualité du point de vue exécution [Temps de réponse, disponibilité, fiabilité], sécurité [authentification, confidentialité, non-répudiation,..] et la qualité métier [Coût, réputation].

6.1.2. Les concepts statiques: Les concepts statiques sont utilisés pour décrire les propriétés (les paramètres) des services web, par exemple, les E/Ss d'une opération doivent avoir une représentation sémantique qui explicite leurs liens de subsomption afin de pouvoir vérifier si deux opérations ayant des E/Ss différents sont similaires pour être substituées ou compatibles pour être composées ensemble. Pour cela, une

ontologie des concepts est fournie pour chaque domaine. L'hiérarchie dans l'ontologie explicite les liens de subsomption entre les concepts les plus spécifiques à leurs concepts génériques.

6.2. Schéma de mise en correspondance

La mise en correspondance sémantique est un mécanisme qui vise à trouver les similarités sémantiques et éventuellement structurelles entre deux types d'informations: l'information recherchée et l'information publiée et ceci en se basant sur un support sémantique. Nous définissons pour le matching sémantique une fonction "*match*" qui vérifie si deux concepts syntaxiquement différents appartenant à une même ontologie sont sémantiquement similaires ou compatibles. Dans notre cas, nous appliquons la fonction "*match(x,y)"* pour calculer la relation de subsomption entre deux paramètres x,y de deux opérations différentes en utilisant l'ontologie du domaine dans laquelle ils sont définis. Cette fonction "*match*" se base sur le principe de matching de Paolucci [12] pour calculer le lien de subsomption entre les concepts. Cependant nous l'avons utilisé dans un contexte plus flexible, dans le sens où il n'est pas nécessaire que toutes les E/Ss du service recherché soient mises en correspondance avec toutes les E/Ss du service publié. Nous avons besoin uniquement de faire correspondre toutes les sorties du service recherché à toutes ou quelques sorties du service publié (car on peut accepter un service qui produits des résultats en plus), et faire correspondre toutes ou quelques entrées du service recherché à toutes les entrées du service publié (car on suppose qu'un service donné ne peut s'exécuter si l'une de ses entrées n'est pas disponible). Nous définirons par la suite les règles de correspondance concernant les paramètres d'E/Ss, mais avant cela, nous décrirons les modes de mise en correspondance identifiés:

a) **Mise en correspondance horizontale :** elle est appliquée lorsqu'on a besoin de vérifier si deux opérations OP1 et OP2 peuvent être composées ensemble (enchaînées). Ceci consiste à faire correspondre les sorties de la première opération aux entrées de la deuxième (Fig ci-après).

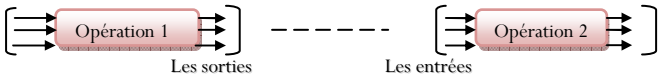

Figure 4. *Mise en correspondance horizontale*

b) **Mise en correspondance verticale 1:1:** est nécessaire lorsqu'on veut vérifier si deux opérations OP1 et OP2 ont des fonctionnalités similaires, c'est-à-dire, si l'opération OP1 peut être substituée par l'opération OP2 (Fig ci-après)..

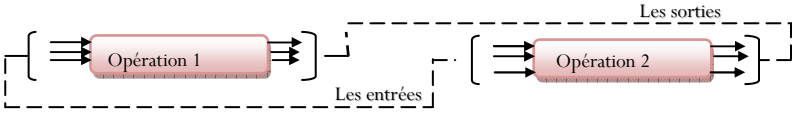

Figure 5. *Mise en correspondance verticale 1:1*

c) **Mise en correspondance verticale 1:N :** est appliquée lorsqu'on veut vérifier si une opération OP1 peut être substituée par une chaine d'opérations simples tel que les entrées de l'opération recherchée (OP1) correspondent sémantiquement aux entrées de la première opération de la chaîne et ses sorties correspondent sémantiquement aux sorties de la dernière opération de la chaîne (Fig ci-après)..

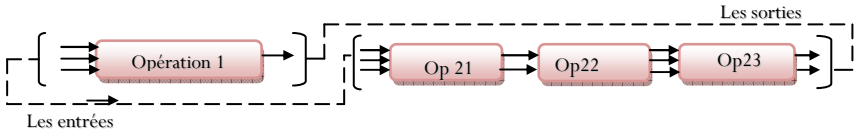

Figure 6. *Mise en correspondance verticale 1:N*

Nous définissons les règles de correspondance entre les paramètres d'E/Ss comme suit :

∗ **Les sorties:** rappelons que nous avons besoin de faire correspondre toutes les sorties du service recherché à toutes ou quelques sorties du service publié:

a) Exact: $Out(OP_R) == Out(OP_P)$ si :
- $Out(OP_R)$ et $Out(OP_P)$ ont le même nombre de variables et
- $\forall x \in Out(OP_R)$ (resp. $Out(OP_P)$),$\exists y \in Out(OP_P)$ | *match(x,y)= "exact "* .

b) Plug-In: $Out(OP_R) =\approx Out(OP_P)$ si :
- $Out(OP_R)$ et $Out(OP_P)$ ont le même nombre de variables de sortie et
- $\forall x \in Out(OP_R)$,$\exists y \in Out(OP_P)$ | *match(x,y)= "exact "* ou *match(x,y)= "plug-in"* et $\exists z \in Out(OP_R)$,$\exists t \in Out(OP_P)$ | *match(z,t)= "plug-in"*

c) Inclusion exacte: $Out(OP_R) \subset Out(OP_P)$ si:
- $Out(OP_P)$ contient plus de variables que $Out(OP_R)$
- $\forall x \in Out(OP_R)$,$\exists y \in Out(OP_p)$ | *match(x,y)= "exact "* .

d) Inclusion Plug-In: $Out(OP_R) \subset\approx Out(OP_P)$ si :
- $Out(OP_P)$ contient plus de variables que $Out(OP_R)$.
- $\forall x \in Out(OP_R)$,$\exists y \in Out(OP_P)$ | *match(x,y)= "exact "* ou *match(x,y)= "plug-in"* et $\exists z \in Out(OP_R)$,$\exists t \in Out(OP_P)$ | *match(z,t)= "plug-in"*

Rappelons que d'après **[12]**:
- *match(x,y)= "exact"* si x, y font référence à la même classe dans l'ontologie.

- *match(x,y)= "plug-in"* si classe de x dans l'ontologie est une sous classe de y. On dit dans ce cas que le concept publié y subsume x c'est à dire que y est plus générale que le concept recherché x.
- *match(x,y)= "subsumé"* signifie que classe de x dans l'ontologie est une sous classe de y. On dit dans ce cas que le concept publié y *subsume* le concept recherché x.
- *match(x,y)= "disjoint"* si aucune relation de subsumption lie x et y dans l'ontologie.

Les deux cas qui peuvent fournir un résultat pertinent sont le cas *"exact"* et le *"plug-in"*. Le cas "subsumé" n'est pas pris en considération car il fournit un matching non complet.

∗ Les entrées:

Pour les entrées, nous avons besoin de faire correspondre toutes ou quelques entrées du service recherché à toutes les entrées du service publié. Par conséquent, nous définissons $In(OP_R) == In(OP_P)$ (exact), $In(OP_R) =\approx In(OP_P)$ (plug-in) de façon similaire que $Out(OP_R) == Out(OP_P)$, $Out(OP_R) =\approx Out(OP_P)$ et les autres cas comme suit :

c) Inclusion exacte: $In(OP_R) \supset In(OP_P)$ si:

- $In(OP_R)$ contient plus de variables que $In(OP_P)$
- $\forall x \in In(OP_P), \exists y \in In(OP_R)$ | *match(y,x)= "exact "* .

d) Inclusion Plug-In: $In(OP_R) \approx\supset In(OP_P)$ si :

- $In(OP_R)$ contient plus de variables que $In(OP_P)$.
- $\forall x \in In(OP_P), \exists y \in In(OP_R)$ | *match(y,x)= "exact "* ou *match(y,x)= "plug-in"* et $\exists z \in Out(OP_R), \exists t \in In(OP_R)$ | *match(z,t)= "plug-in"*

∗ Les sorties avec les entrées:

Dans ce cas (la mise en correspondance horizontale), nous avons besoin de mettre en correspondance toutes ou quelques sorties du premier service avec toutes les entrées du service suivant. Nous obtenons alors 4 cas de correspondance: Exact ($Out(OP_1) == In(OP_2)$), Plug-in ($Out(OP_1) =\approx In(OP_2)$), Inclusion exacte ($Out(OP_1) \subset In(OP_2)$) et Inclusion plug-in($Out(OP_1) \subset\approx In(OP_2)$).

Ainsi pour chaque niveau et selon la combinaison de l'ensemble des entrées avec l'ensemble des sorties, nous distinguons 4 niveaux de correspondance pour chaque mode de correspondance: exact, plug-in, inclusion exacte et inclusion plug-in.

6.3. Cycle de vie de la requête de composition

Afin de permettre à un utilisateur d'accéder graduellement, à travers une requête simple, aux fonctionnalités complexes offertes par une combinaison de services web, nous proposons différents niveaux de modélisation du plan de composition. Ces derniers constitueront les différentes phases de résolution de la requête de composition (voir la figure 7)

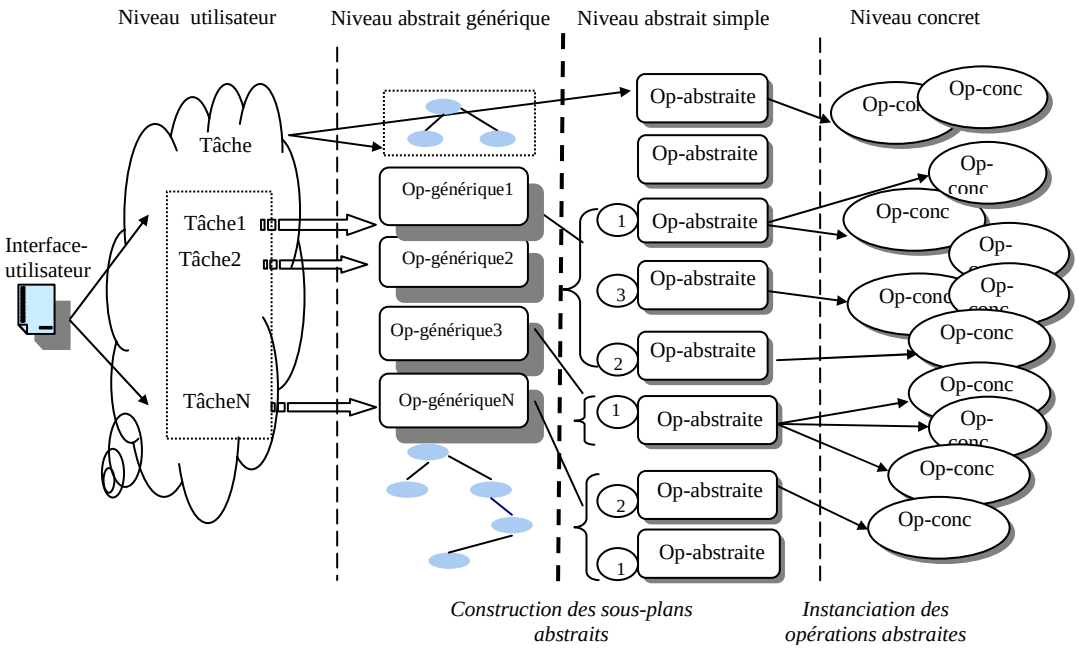

Figure 7. *Les différents niveaux de mise en correspondance.*

- **Requête utilisateur :** L'utilisateur peut ne pas avoir une idée complète et assez précise concernant le service qu'il cherche. Ainsi, la précision d'une formulation de requête peut varier d'un utilisateur à un autre. Dans le cas général, nous considérons la requête comme étant composée d'une partie fonctionnelle: Nom du domaine sur lequel il veut faire sa recherche, une ou plusieurs sous-tâches avec leurs noms de paramètres d'E/Ss, une valeur entière entre 0 et 1 représentant le seuil de correspondance qu'il précise lui même, et une partie non-fonctionnelle représentée par *préférences* dans lequel l'utilisateur spécifie un poids Wi[0,1] pour chaque attribut de qualité QdWS.

$$Requête :- nomDomaine, \cup_{i=1}^{n} Sous\text{-}tâche_i (Nom,In,Out) , seuil, \cup_{j=1}^{k} , préférences$$

- **Template générique :** Elles sont considérées comme des processus métiers connus dans un domaine d'application qui correspondent à des besoins communs des utilisateurs en terme de processus complexe. Dans notre cas, une template générique est la combinaison de plusieurs opérations génériques au sein d'un même processus métier suivant un modèle d'orchestration. Elles seront, donc, utilisées pour définir un plan initial de composition et permettent d'offrir une vue générale et simplifiée d'un processus métiers complexes du domaine d'application. Citons des exemples de processus très souvent sollicités qui peuvent être modélisés grâce à des templates génériques: organisation d'un voyage (Figure 7.), achat de véhicule… et.

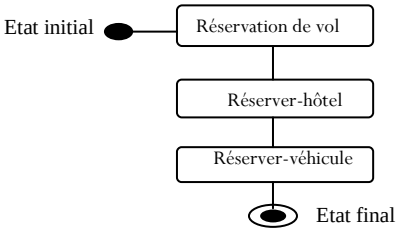

Figure 8. *Orchestration d'une Template générique grâce à un diagramme d'activité*

Si la requête de l'utilisateur est : $\cup_{i=1}^{n}$ *Sous-tâche*$_i$ *(Nom,In,Out)tel que n !=0.*
Nous appliquons une mise en correspondance verticale 1:1 entre chaque sous-tâche (notée par OP_R) spécifiée dans la requête et chaque opération générique(OP_{TG}) appartenant à une même template générique (voir la figure 9).

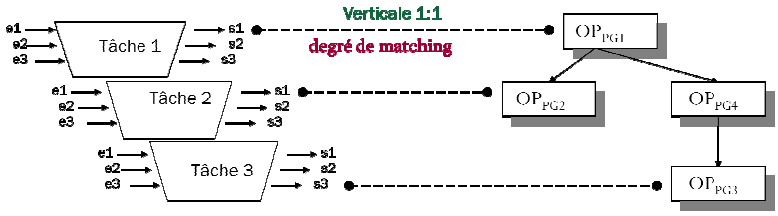

Figure 9. *La mise en correspondance de la requête à une Template générique*

Selon les règles de correspondance définies, nous obtenons, des niveaux de correspondance quantifiés par une valeur que nous notons par : *degré (OP_{Ri} : OP_{TGi})*. Nous quantifions le degré de correspondance totale de la requête comme suit :

$$Degré\text{-}corresp\ (Q) = [\Sigma^{n}_{i=1} degré\ (OP_R i : OP_{TG} i)/n] * \begin{cases} 1 & si\ \ nb(OP_R)=nb(OP_{TG})=n \\ ½ & si\ \ nb(OP_R)<nb(OP_{TG}) \end{cases}$$

Deux types de résultats sont acceptables :

1) Le nombre des sous-tâches formant la requête est égale au nombre d'opérations génériques formant la template générique, on dit dans ce cas que la requête est *sémantiquement équivalente* à la template générique
2) Le nombre des sous-tâches formant la requête est strictement inferieur au nombre d'opérations génériques formant la template générique, on dit que la template couvre le besoin de la requête

- **Le plan abstrait de composition :** Représente le deuxième niveau de traitement de la requête de composition. Un plan abstrait de composition est une agrégation d'un ensemble d'opérations abstraites suivant un modèle d'orchestration y compris le flux de données. Le passage d'une Template générique au plan abstrait se fait grâce à une mise en correspondance de chaque opération générique avec une opération abstraite ou une combinaison d'opérations abstraites suivant les règles de correspondance (verticale 1 :1 ou 1 :N et horizontale) définies auparavant, voir la figure suivante.

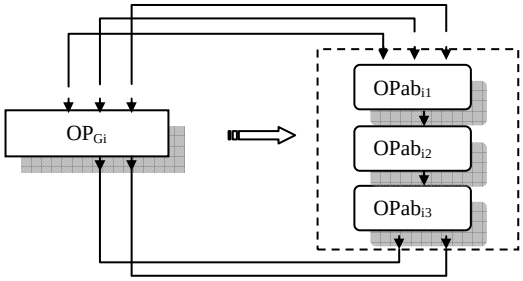

Figure 9. *Construction d'un sous plan abstrait suivant les règles de correspondance*

La deuxième étape consiste à remplacer chaque opération générique dans la template générique (TG) par le sous plan correspondant (SPab) voir la figure 11:

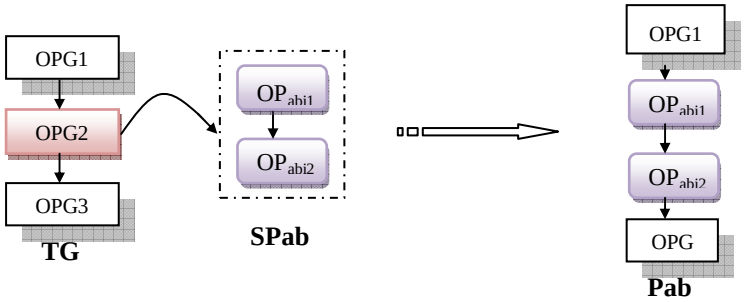

Figure 10. *Remplacement d'une opération générique par son sous-plan abstrait*

- **Le plan concret de composition (le plan exécutable):** Un plan concret de composition est une agrégation d'un ensemble d'opérations concrètes suivant un modèle d'orchestration définissant le flux de contrôle et le flux de données entre les opérations participantes. Il est généré automatiquement à partir du plan abstrait. Chaque opération abstraite du plan abstrait sera instanciée en une opération concrète grâce aux règles de correspondance verticale 1 :1. Plusieurs plans concrets peuvent être générés car plusieurs opérations concrètes peuvent implémenter la même fonctionnalité décrite par une opération abstraite. Pour cela, la mise en correspondance entre une opération abstraite et une opération concrète considère d'autres informations concernant l'implémentation et la qualité d'une opération (ex: le coût, le temps de réponse..). Cependant, Le problème qui se pose est le suivant : comment trouver un plan concret optimal (qui répond au mieux aux préférences de l'utilisateur) en un temps raisonnable? Ce problème a été particulièrement traité par **[11].** Nous avons adapté les résultats de son travail à notre modèle. En effet **[11]** propose deux manières pour générer le plan concret exécutable:

Recherche exhaustive: cette stratégie consiste à générer tous les plans concrets possibles, les évaluer en terme de qualité de service puis sélectionner le meilleur pour être exécuté.

L'évaluation des plans concrets en terme de qualité se fait en calculant la qualité de service (QdWS) de chaque plan concret généré de la manière suivante:

Si on suppose que la qualité de service d'un service concret quelconque est représentée par le vecteur suivant : QdWS(SCi)=(T-reponse(SCi) (le temps de réponse du servicei), Dispo(SCi)(sa disponibilité), Fiab(SCi) (sa fiabilité), Coût(SCi) (le coût en dollars), Réput(SCi) (sa réputation), Auth(SCi)(si le service doit être authentifié), Confid(SCi) (sa confidentialité), Répud(SCi) (sa répudiation)).Les paramètres QdWS du plan entier est calculé par la fonction d'agrégation de la manière suivante (voir le tableau ci-dessous):

QdWS	Fonction d'agrégation
Temps de réponse (T-reponse) [réel]	$\sum_{SWi \in \text{ long-path}}$ T-reponse(SCi)
Coût (Coût) [réel]	$\sum_{i=1}^{n}$ Coût(SCi)
Disponibilité (Dispo) [0,1]	$\prod_{i=1}^{n}$ Dispo(SCi)
Réputation (Réput) [0,1]	$1/n\sum_{i=1}^{n}$ Réput(SCi)
Authentification (Auth) {0,1}	$1/n\sum_{i=1}^{n}$ Auth(SCi) ou $\prod_{i=1}^{n}$ Auth(SCi)
Confidentialité (Confid) {paramètres à ne pas divulguer}	$\cup_{i=1}^{n}$ Confid(SCi)
Non-répudiation (Répud) {0,1}	$1/n\sum_{i=1}^{n}$ Répud(SCi) ou $\prod_{i=1}^{n}$ Répud (SCi)
Fiabilité (Fiab) [0,1]	$1/n\sum_{i=1}^{n}$ Fiab(SCi) ou $\prod_{i=1}^{n}$ Fiab (SCi)

Remarque: Le temps de réponse du plan entier est celui du plus long chemin dans le plan (long-path).
Le plan qui sera sélectionné pour l'exécution est donc celui qui possède la meilleure combinaison des valeurs obtenues par la fonction d'agrégation des différents paramètres de QdWS qui peuvent satisfaire au mieux les préférences de l'utilisateur.

L'avantage d'une telle stratégie est qu'elle garantit de trouver la meilleure solution (c'est à dire, le meilleur plan) en explorant toutes les solutions possibles. Cependant, l'inconvénient de cette stratégie est évident: la complexité de calcul est d'ordre $\theta(M^N)$ ce qui est inapplicable si M (nombre maximum d'opérations concrètes par opération abstraite) et N (le nombre d'opération abstraite dans le plan abstrait) sont très grands.

Dans la stratégie de la sélection locale, la meilleure opération concrète est sélectionnée pour chaque opération abstraite au moment de la construction du plan concret. La mise en correspondance, l'évaluation et la sélection sont faites en une seule étape. Dans cette stratégie, l'évaluation est effectuée en utilisant une fonction appelée *fonction objective* qui permet de décider quelle est la meilleure opération concrète parmi les M opérations candidates qui va être choisie pour implémenter l'opération abstraite en cours. La *fonction objective* (F) de chaque opération (OP_c) est calculée de la manière suivante:

$$F(OP_c) = md(OP_c) * (P_{neg} + P_{pos}) \text{ où}$$
$$P_{neg} = \Sigma_{neg}\, Wi\, (pQ_{imax} - pQ_i / pQ_{imax} - pQ_{imin})$$
$$P_{pos} = \Sigma_{pos}\, Wi\, (pQi - pQ_{imax} / pQ_{imax}\, x - pQ_{imin})$$

- md(*OPc*) définit le degré de Matching calculé lors de la mise en correspondance de l'opération abstraite en cour à l'opération concrète OPc.
- pQ_i^{max} est la valeur maximum du i^{eme} paramètre par rapport à toutes les opérations concrètes candidates (implémentant l'opération abstraite en cours) et le PQ_i^{min} est le minimum.
- *Neg et pos*: désignent respectivement les paramètres de qualité négatifs et positifs. Un paramètre négatif est un paramètre à minimiser tels que le coût, temps de réponse et un paramètre de qualité positif est un paramètre dont la valeur est à maximiser telles que la réputation, disponibilité…etc

La meilleure opération concrète qui sera choisie pour implémenter l'opération abstraite en cours est celle possédant la meilleure valeur de la fonction objective ($F(OP_c)$).
Cette stratégie est plus réaliste et moins coûteuse en temps. Cependant, elle peut rater la solution optimale.

- **Spécification détaillée et exécution du service composite:** Cette dernière phase consiste à générer une description détaillée du service composite en utilisant un langage de spécification de flux tel que BPEL4WS **[2]** ou autres. Cette description doit comprendre une définition des services participants, les messages échangés, leurs paramètres et le flux de contrôle et de données entre eux. Cette spécification sera soumise directement à un moteur d'exécution de workflow tel que le BPWS4J ou le BPEL Orchestration Server **[1]**.

7. Tableau comparatif

Afin de mieux positionner notre proposition par rapport aux autres approches de composition des services web étudiées, nous proposons ce tableau comparatif (tableau ci-dessous). Ce dernier permet une comparaison selon certains critères d'évaluation que nous avons pu déduire à travers notre étude.

	Méthodes manuelles	**Méthodes à base de workflow**	**Méthodes à base de planification**	**Méthodes interactives**	**L'approche proposée**
Niveau d'automatisation	Aucune automatisation du processus de composition (nul)	Seule la recherche et la liaison aux services concrets est automatique (moyen)	Pratiquement tout le processus se fait de manière automatique (élevé)	Semi-automatique	Assez élevé : l'utilisateur peut intervenir uniquement pour sélectionner un template
Niveau de dynamicité	Tout se fait statiquement	-Modèle de processus est généré statiquement -Liaison aux services concrets est dynamique	- la génération du plan de composition se fait de manière dynamique	- la génération du plan se fait de manière dynamique	Moyen : Les templates génériques sont statiquement construits. La génération du plan abstrait et concret est dynamique.
Niveau de Scalabilité	Pas du tout	Moyen	élevé	Modeste.	Assez élevé: pratiquement, toutes les étapes du processus de génération du plan se fait de manière automatique
Formulation de la requête par	Très bas niveau,	Bas niveau : l'utilisateur doit	Définit l'état initial, l'état final et	Se fait interactivemen	Langage simple et ne nécessite pas des détails techniques :

l'utilisateur	c'est l'utilisateur qui définit tout le plan de composition	mettre en place le modèle de processus en utilisant un langage de flux	certaines contraintes avec un langage de très bas niveau (les langages logiques) ce qui est très contraignant	t avec le système (au fur et à mesure)	spécifier les tâches qu'il veut exécuter et ses préférences en utilisant les termes de son vocabulaire
Niveau de granularité des plans de composition générés	Un seul plan généré (plan exécutable)	-Workflow abstrait (mais défini manuellement) -Et un workflow exécutable	Un seul plan (plan exécutable) : réutilisabilité très réduite	Uniquement le plan exécutable	Deux niveaux de granularité : plan abstrait et plan concret implique un haut niveau de réutilisabilité
Support sémantique et niveau d'abstraction	Pas de support sémantique (registre UDDI)	La majorité utilise des descriptions syntaxiques (WSDL, BPEL4WS, WSFL…)	Utilise les formalismes logiques, DAML-S, OWL-S.	Support sémantique : DAML-S, OWL-S	Support sémantique qui permet un accès progressif aux fonctionnalités offertes (représentation qui fera l'objet d'une extension du OWL-S).
Recherche des services	Primitif (dans les registres UDDI)	La majorité utilise le moteur de recherche UDDI	dans les profiles DAML-S ou OWL-S	Dans les profiles DAML-S ou OWL-S	Deux niveaux de recherche: - opérations simples - opération générique (template générique)

8. Conclusion et perspectives

Le modèle proposé permet une résolution progressive de la requête de composition. En effet, l'utilisateur spécifie simplement ce qui doit être fait en terme de tâches qui doivent être effectuées et non pas en termes de services qu'il doit invoquer. La découverte des services participants et leur composition sont effectuées de façon automatique et transparente vis-à-vis de l'utilisateur grâce à un schéma de mise en correspondance flexible. Ce schéma définit différents niveaux de correspondance (exact, plug-in, inclusion, …) permettant d'accepter des résultats qui ne sont pas nécessairement identiques à ce qui est demandé mais qui restent pertinents vis-à-vis du besoin recherché.

Vu la complexité du système de composition, plusieurs extensions peuvent être envisagées à ce travail notamment: Enrichir la description sémantique par de nouvelles informations telles que les pré-conditions et post conditions et rendre ainsi plus intelligent le schéma de matching, prendre en compte les besoins non communs aux

utilisateurs (et donc non-prévus par le système). Nous envisageons aussi de développer un processus de résolution de la requête de l'utilisateur et d'optimisation du plan de composition généré selon les préférences des utilisateurs en termes de contraintes de qualité locales et globales. Et enfin, à présent nous étudions les possibilités d'étendre le langage OWL-S pour supporter la représentation sémantique proposée et mettre en place par la suite un protocole de publication des services web selon notre Framework.

8. Références Bibliographiques

[1]: *"Business Process Modeling Language"*. URL: http://www.bpmi.org/

[2]: BEA, IBM, Microsoft (2003) *"Business Process Execution Language for Web Services (BPEL4WS)"*. http://xml.coverpages.org/bpel4ws.html.

[3]: F. Casati, S. Ilnicki, and L. Jin. *"Adaptive and dynamic service composition in Eflow"*. In Proceedings of 12th International Conference on Advanced Information Systems Engineering (CAiSE), Stockholm, Sweden, June 2000.

[4]: Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., and Wilkins, D. *"PDDL: the planning domain definition language"*. In AIPS-98 Planning Committee (1998).

[5]: S. McIlraith and T. C. Son. *"Adapting Golog for composition of Semantic Web services"*. In proceedings of the 8th International Conference on Knowledge Representation and Reasoning(KR2002), Toulouse, France, April 2002.

[6]: B . Medjahed, A. Bouguettaya, and A. K. Elmagarmid. *"Composing Web services on the Semantic Web"*. The VLDB Journal, 12 (4), November 2003.

[7]: Tarek MELLITI, *"Interopérabilité des services Web complexes, Application aux systèmes multi-agents"*, thèse de Doctorat de l'Université Paris IX Dauphine, décembre 2004.

[8]: S. Narayanan and S. McIlraith. *"Simulation, verification and automated composition of Web service"*. In Proceedings of the 11th International World Wide Web Conference, Honolulu, Hawaii, USA, May 2002. ACM.

[9]: http://www.w3.org/TR/owl-features/

[10]: http://www.ai.sri.com/daml/services/owl-s/1.2/

[11] Ouzzani, M., "Efficient Delivery of Web Services", PhD Thesis, Virginia Polytechnic, (2004).

[12]: Paolucci, M., Kawamura, T., Payne, T., Sycara, K., "*Semantic Matching of Web Services Capabilities*", in Proc. of Intl. Semantic Web Conference, Sardinia, Italy, pages 333–347. June 2002.

[13]: S. R. Ponnekanti and A. Fox. *"SWORD: A developer toolkit for Web service composition"*. In Proceedings of the 11th World Wide Web Conference, Honolulu, HI, USA, 2002.

[14]: Rao, P. K¨ungas, and M. Matskin. *"Application of Linear Logic to Web service composition"*. In Proceedings of the 1st International Conference on Web Services, Las Vegas, USA, June 2003.

[15]: Jinghai Rao and Xiaomeng Su, *"A Survey of Automated Web Service Composition Methods" Norwegian University of Science and Technology Department of Computer and Information Science* N-7491, Trondheim, Norway, 2004.

[15]: W3C (2003), *"Simple Object Access Protocol (SOAP)"*. http://www.w3.org/TR/soap.

[17]: W3C (2003), "*Universal description, discovery, and integration (UDDI)"*. http://www.uddi.org

[18]: D. Wu, E. Sirin, J. Hendler, D. Nau, and B. Parsia. *"Automatic Web services composition using SHOP2"*. In *Workshop on Planning for Web Services*, Trento, Italy, June 2003.

[19]: R.Waldinger. "*Web agents cooperating deductively"*. In Proceedings of FAABS 2000, Greenbelt, MD, USA, April 5–7, 2000, volume 1871 of Lecture Notes in ComputerScience, pages 250–262. Springer-Verlag, 2001.

[20]: W3C (2003), "Web Services Description Language (WSDL)". http://www.w3.org/TR/wsdl.