Un Système de Programmation Fonctionnelle pour la Composition de Services Web

CHOUIREF Zahira[1], BELKHIR Abedelkader [2]
[1] Université M'Hammed Bougara - Boumerdes -Algérie
[2] Laboratoire des systèmes informatiques USTHB -Algérie
[1]zahirachouiref@gmail.com
[2]Kaderbelkhir@hotmail.com

Résumé : La troisième génération du web est orientée service. Cette orientation favorise l'interopérabilité des applications et des systèmes. Cependant, la découverte et l'utilisation des services restent explicites. Pour cela, on s'intéressera à la description, la découverte et l'utilisation des services Web sémantiques d'une manière automatique. Cet article décrit la conception d'un système de programmation fonctionnelle capables d'exprimer les services Web (basés XML), et de pouvoir les utiliser par d'autres utilisateurs ou applications. L'approche proposée modélise les services comme des fonctions possédant une signature décrite par un nom, les entrées et les sorties et leurs compositions utilisant le principe de la composition de fonctions en se basant sur un mécanisme d'appariement sémantique. Ce dernier exploite les informations sémantiques des services de type 'offre' et des services de type 'demande' représentés dans des structures de données sémantiques sans prendre en considération les besoins non-fonctionnels de l'utilisateur.

Mots clé: Service Web, composition de services Web sémantiques, fonction, composition de fonctions.

Introduction

L'explosion de l'Internet a profondément changé nos méthodes de travail. Les logiciels informatiques sont alors devenus omniprésents dans les entreprises, les agences, les hôpitaux, les écoles, et sont de plus en plus utilisés dans la vie quotidienne (par exemple, téléphone, ordinateur portable, PDA, voiture, micro-onde et achat en ligne). L'avènement de nouveaux usages de l'informatique dite mobile et ambiante ne permet plus de concevoir des applications logicielles dédiées à des plates-formes prédéfinies, composées d'un ensemble de dispositifs qui sont a priori connus.

La troisième génération du Web est orientée service [3]. Cette orientation favorise l'interopérabilité des applications et des systèmes. Les architectures à base de services (SOA) consistent à concevoir des applications distribuées à l'aide de composants réutilisables et interopérables. La propriété d'interopérabilité des services est leur capacité à s'invoquer et à se répondre mutuellement indépendamment des plates-formes sous-jacentes et des langages dans lesquels ces services ont été développés. Lorsque les interactions s'appuient sur Internet et sur les standards Web, on parle de services Web [4]. Les services Web sont des technologies émergentes et prometteuses pour le développement, le déploiement et l'intégration d'applications Internet. Actuellement, les services Web sont mis en œuvre à travers les trois standards : WSDL, UDDI et SOAP [7], [8]. Ces protocoles facilitent la description, la découverte et la communication entre services.

L'accès aux services Web est défini comme étant la succession de trois étapes, à savoir : la recherche de fournisseurs du service souhaité, la sélection de l'un de ces fournisseurs et la réalisation du service par l'invocation du fournisseur choisi. Cependant, les services Web ne sont pas capables de résoudre tous les problèmes. Actuellement, de nombreuses infrastructures pour supporter des services, sont déployées par différents organismes [6]. La diversité de ces infrastructures et des organismes qui les déploient entraîne des hétérogénéités. En effet, la réponse aux besoins complexes des utilisateurs peut correspondre à l'emploi de plusieurs services hétérogènes (simples et/ou composites) exécutés conjointement. Ce type de problème est connu comme la composition de services web. Différentes approches ont été citées, pour répondre à cette problématique de compositions [9], [10], [11]. Elles diffèrent par leur degré d'automatisation (manuelles, automatiques), par leur caractère dynamique vs statique, ie est ce que le plan de service est créé à priori ou bien durant l'exécution du système, et par leur prise en charge de la sémantique (basées sur les ontologies ou non).

L'interprétation consistante des données échangées entre services Web composés est gênée par des différences de représentation et d'interprétation sémantiques.

Dans ce contexte, nous proposons un système **FS4WSC** (*Functional System For Web Service Composition*) (voir section 3) pour composer automatiquement les services web sémantiques (SWS).

L'article est structuré comme suit : la section 2 présente quelques concepts sur lesquels se base notre réflexion. La section 3 décrit notre approche, où nous présentons l'architecture globale du système proposé pour faciliter la découverte et la composition de SWS dans laquelle nous retrouvons essentiellement la plate-forme de services Web, un système d'annotation sémantique qui utilise une ontologie du domaine, un module de correspondance et un moteur de composition pour

l'appariement entre la requête et les services disponibles et la composition des services si c'est nécessaire. Ensuite, nous présentons notre approche "*Approche par les fonctions*" où nous proposons un formalisme pour décrire les aspects syntaxiques et sémantiques de la requête et des SWS, puis nous décrivons les algorithmes d'appariement sémantique, de découverte et de composition de SWS. Une étude de cas est présentée dans la section 4 où nous utilisons le langage de *programmation fonctionnelle* CAML [5] pour mettre en œuvre les algorithmes proposés et les appliquer au domaine de réservation de voyages en ligne. Finalement, dans la section 5 une conclusion résume le travail et discute des perspectives pour son amélioration.

1. Type, concept et fonction

D'après Preeda Rajasekaran [2], faire un mapping entre les concepts sémantiques utilisant une ontologie pour une meilleure intégration de données nécessite dans des cas d'appliquer des fonctions de mapping et de transformation de type, car c'est souvent difficile de trouver des équivalences entre les concepts ontologiques et les données dans les applications. L'auteur a étudié la notion de théorie des types et les langages fonctionnels et leurs influences sur le domaine des SWS, ainsi qu'il a montré l'importance des types et des systèmes de types où l'utilisation des propriétés offertes de ces deux champs peuvent être adaptées pour rehausser les caractéristiques des systèmes de types comme offertes par les ontologies, c-à-d inclure l'inférence et la transformation de types.

Les types font référence aux types de données tel que int, flaot… etc dans les langages de programmation, et aux classes dans l'ontologie. Les classes de l'ontologie représentent les types des différents concepts d'un domaine donné. Les types sont utilisés pour désambigüiser entité/objet/concept d'après ses propriétés, fonction et relations. Les types de bases font références aux objets individuels, les types complexes font référence aux classes, les propriétés et relations sont exprimées comme fonctions qui rentrent des arguments (propriétés) et rendent une valeur booléenne.

Un système de types [1] aide à classer les valeurs d'un programme en ensembles appelées "types", donc il met en vigueur des restrictions pour développer des programmes bien formés. Ce trait est essentiel dans la détection des erreurs lors de la compilation. Appliquer les concepts de système de types aux ontologies, permet de tirer une analogie. Une entité dans une application qui est annotée avec un concept ontologique devrait satisfaire toutes les propriétés associées à ce concept. Si cette condition n'est pas satisfaite, il faut appliquer les règles de transformation pour obtenir le mapping exigé. Un validateur pour annotation (semblable aux compilateurs pour les langages de programmation) devrait assurer ceci.

2. Proposition d'une approche pour la composition automatique des services Web - Approche par les fonctions -

L'étude des travaux existants a montré qu'à ce jour différentes dimensions de composition des services web font l'objet d'étude. Cependant, ce problème de composition automatique de services Web est par nature très difficile. En effet les données sont instables, le Web étant dynamique.

Notre principale contribution est de proposer un système complet pour la composition, car nous considérons qu'une composition de services web doit s'engager dès la découverte de services jusqu'à l'interaction avec les utilisateurs. Un deuxième point sur le système proposé est que la composition peut être faite d'une façon transparente. Du point de vue de l'utilisateur, une fois que la requête est définie, le système s'engage à composer les services nécessaires, et lui proposer à la fin les compositions trouvées. Afin d'atteindre ces objectifs, nous proposons **FS4WSC** *(Functional System For Web Service Composition)*, un système pour composer automatiquement les services web sémantiques (SWS) basé sur une description d'un modèle dans le contexte de la composition des services web.

Dans cette partie, nous soulignons tout d'abord que notre travail se base sur une structure générale représentant l'ensemble des services Web disponibles, modélisés sous forme de *fonction*. Dans un second temps, nous précisons que la base de notre travail est les services web découverts issus de la requête de client et leurs compositions afin de satisfaire les besoins du client.

2.1. Architecture du système

L'utilisation de l'architecture classique des services Web et l'intégration du formalisme de la sémantique ne modifie pas le système existant et facilite la mise en œuvre de cette architecture. Pour cela, nous présentons d'abord l'architecture proposée qui est illustrée dans la **Figure 1**. Les différents sous-systèmes qui composent notre architecture sont décrits comme suit :

Architecture de référence qui est basée sur les trois entités. Le fournisseur de services (prestataire), l'annuaire de services et le client (utilisateur du service).
Système d'annotation sémantique. Notre contribution réside dans l'ajout, à cette architecture, un composant dédié à l'annotation des services Web. Le système d'annotation identifie d'abord automatiquement dans un service les éléments pertinents à annoter, puis il exploite une ontologie pour déterminer quels sont les concepts les plus spécifiques possibles de l'ontologie dont l'instance sera utilisée pour annoter chacun de ces éléments.

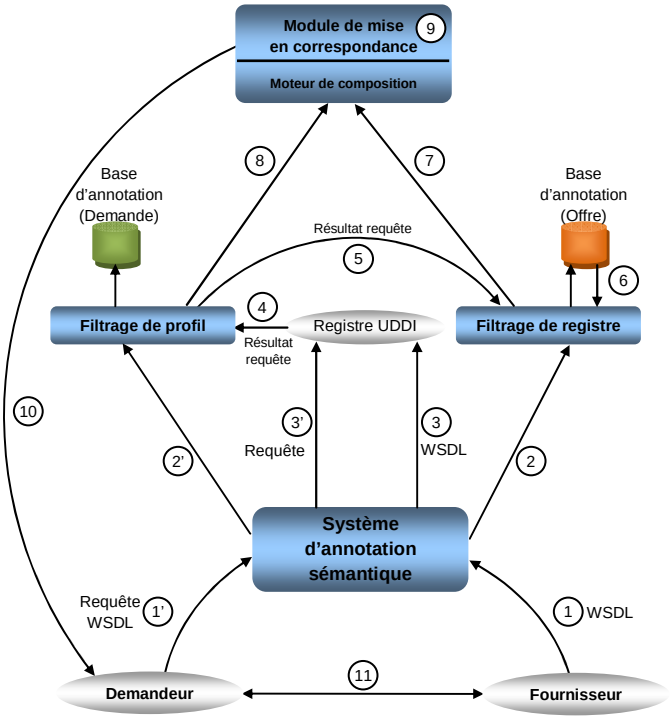

Figure 1 : Proposition d'architecture

Module de mise en correspondance et le moteur de composition. Ce sous système est défini pour permettre la recherche de services Web sémantiques adéquats à la requête. Le module de mise en correspondance effectue les correspondances entre la description d'un SWS et la demande de ce dernier en renvoyant au demandeur un fichier XML en résultat si la requête est satisfaite par un seul service Web. Le processus de correspondance s'appuie sur des règles qui exploitent l'ontologie, les profils des services et la requête du client. Dans le cas où la requête ne peut pas être satisfaite par un seul service, une composition de SWS retournés par le module de mise en correspondance est faite à l'aide du moteur de composition. Voici un scénario type d'utilisation de l'architecture :
(1) Le fournisseur transmet la description du service Web sémantique. Un système d'annotation annote sémantiquement les noms et les entrées/sorties des demandes et des offres de services par des concepts ayant une description formelle afin d'être interprétables par une machine. (2) Les services Web annotés sont envoyés à un composant de filtrage qui extrait la description sémantique des services et il les stocke dans une base d'annotation (Offre). (3) La description classique WSDL est ensuite transmise au registre UDDI, comme dans l'architecture de référence. (1') Du côté du demandeur, de la même manière que nous avons expliqué dans (1) et

(2), l'utilisateur soumet sa requête de service au même système d'annotation qui annote la requête puis (2') la soumet à un second composant de filtrage. Celui-ci stocke la description sémantique du service attendu dans une autre base d'annotation (Demande). (3') La requête du service, sans la description sémantique, est ensuite transmise de manière classique au registre UDDI qui procède à l'exécution de la requête. (4) Le résultat de la requête (la liste des services répondant à la demande de l'utilisateur) est intercepté par le module de *filtrage de profil* qui génère un fichier XML comportant les descriptions attendues. (5) La liste des services est communiquée au module de *filtrage de registre* afin qu'il recherche, si elles existent, leurs descriptions sémantiques. (6) Le module de *filtrage de registre* accède à la base d'annotation (Offre) pour rechercher les SWS adéquats, puis génère un fichier XML comportant les descriptions des SWS. Ce fichier XML d'une part (7) et le fichier XML des demandes de l'utilisateur d'autre part (8) sont transmis au module d'appariement. (9) L'appariement de services dans ce module s'effectue à deux niveaux : d'abord, la découverte d'un ensemble de services susceptibles de répondre à la requête du client, puis la sélection du service qui puisse, effectivement, y répondre à la requête. Le moteur de composition accède à la liste des services retournée par le module d'appariement afin de composer les services qui peuvent répondre à la requête si c'est nécessaire. Plusieurs compositions peuvent être avoir lieu. Dans notre travail on s'intéressera à la première composition trouvée. (10) Le résultat, sous la forme d'un fichier XML, comporte la liste des services répondant aux attentes de l'utilisateur est envoyé au demandeur de services. (11) La communication entre le demandeur de service et le fournisseur se fait ensuite de manière traditionnelle par l'intermédiaire du protocole SOAP.

Ainsi et pour soutenir le processus de composition automatique de services web répondant à la requête, nous proposons un formalisme pour représenter les services Web atomiques et composites ainsi que deux algorithmes qui consistent à découvrir une combinaison de services satisfaisant la requête de client et à vérifier l'existence d'une composition de services Web.

2.2. Un formalisme pour représenter les services Web sémantiques

Dans cette partie, nous utilisons l'approche fonctionnelle pour modéliser des services Web, qui sont vus comme des fonctions. Le formalisme recherché doit permettre d'exprimer ces deux aspects : la sémantique des services Web atomiques et la composition des services Web. Pour montrer l'utilité et l'applicabilité de notre proposition, nous avons besoin de quelques définitions préalables :

Service Web: *S* est défini comme triplet (C, I, O) où C le contexte de la spécification de service (mot clé ayant trait au domaine du service), $I = \{ Ii / i > 0 \}$ un ensemble d'entrées de service et $O = \{ Oj / j > 0 \}$ l'ensemble de sorties de service.

Composition sémantique: deux services *Si* et *Sj* pouvant être composés sémantiquement ensemble et dénoté par $S_i \rhd S_j$ ssi $\exists I_k \in I_{Si}, O_l \in O_{Sj} \mid I_k \subseteq O_l$ où I_{Si} et O_{Sj} sont les ensembles d'entrées et sorties de service respectivement, et $\subseteq$ dénote un symbole de matching (appariement) sémantique.

La définition signifie que deux services peuvent se composer si et seulement si les sorties (pas nécessairement tous) d'un service s'apparient sémantiquement avec les entrées de l'autre.

Service Web Atomique: un service *Si* est atomique s'il n'existe pas $S_n \rhd S_m = S_i$.
Spécification d'une offre et d'une demande de services. Les demandes et les offres de services sont décrites dans notre formalisme par des fonctions, où chaque élément représente un nom et entrées/sorties, dans la signature du service. De plus, les noms des services et leurs entrées/sorties peuvent être annotés sémantiquement. La spécification d'une offre ou d'une demande de services avec un langage fonctionnel est illustrée par la **Figure. 2**. La signification de chaque élément est ci-après :

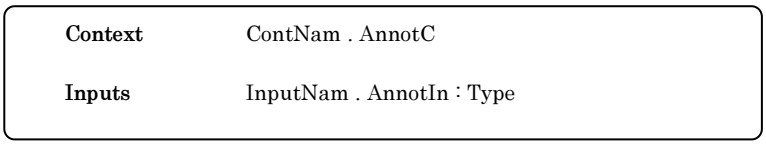

Figure 2 : Spécification d'une offre et d'une demande de services

ContNam décrivant un nom symbolique du service (nom du service) c-à-d nom d'une fonction.

InputNam/OutputNam sont des mots-clés décrivant la liste des variables d'entrées/sorties des services (fonctions dans notre formalisme), respectivement.

AnnotC, AnnotIn, AnnotOut Description formelle des concepts servant à l'annotation sémantique des noms, des entrées et des sorties respectivement des demandes et des offres de services.

Type Définition des types de données abstraits utilisés dans les paramètres d'entrées et de sorties.

Formellement un service Web peut être représenté sous forme d'une fonction *fsi = (Fsi ; Isi ; Osi)*. *Fsi* représente l'ensemble de fonctions (opérations) du service. L'ensemble *Isi* des variables représente les paramètres d'entrées de service et *Osi* son ensemble de paramètres de sorties.

Pour chaque opération, si un tel paramètre d'entrée i doit être fourni pour avoir telle sortie o ceci signifie qu'il existe une dépendance entre *i* et *o*, et pour laquelle une fonction *f(i) = o* est définie dans notre système.

Une fonction composite *fsi o gsj* est définie entre *fsi et gsj*, si et seulement si il y a une similarité de type entre I_k *et* O_l c-à-d $I_k \subseteq O_l$ ($I_k \in Is_i$; $O_l \in Os_j$), où I_{Si} et O_{Sj} sont les ensembles d'entrées et sorties de service *Si* et *Sj* respectivement.

Les structures de données sémantiques. Une opération (fonction) soit celle d'une requête ou bien d'un service, peut posséder une ou plusieurs variables d'Inputs/Outputs, pour cela on utilise la structure de donnée dynamique "Liste". Les "listes" spécifient la structure et la sémantique des informations qui sont communiquées comme paramètres de service dans la composition. Les informations qui sont communiquées à travers les messages doivent être conformes à ces structures. Comme chaque variable a son nom, annotation et type, donc chaque élément de la liste est représenté à l'aide d'une autre structure de donnée statique qui est de type "Record", donc on aura une liste d'enregistrements.

Les structures proposées dans notre travail ont une importance dans la description des capacités des paramètres de services, donc elles forment le fondement pour l'appariement sémantique ainsi que pour la composition.

2.3. Mise en œuvre

La première phase de notre système **FS4WSC** concerne la découverte de SWS: pour cela, nous avons proposé *l'Algorithme 1* de découverte des SWS qui peuvent potentiellement participer dans la composition. La deuxième phase de **FS4WSC** concerne la composition automatique où nous avons utilisé *l'Algorithme 2* de composition pour composer les services découverts dans la première phase s'ils sont composables. La troisième phase de **FS4WSC** qui n'est pas présente dans la liste de nos motivations correspond aux invocations automatiques de services en faisant appel aux services tout en fournissant les entrées et en obtenant les sorties.

Principe de fonctionnement de l'algorithme de découverte (Algorithme. 1). Rappelons que notre problème consiste, dans un domaine particulier, à trouver tous les services et compositions de services qui permettent de répondre à une requête. Pour cela, nous nous appuyons sur les annotations associées aux fonctions et à leurs paramètres d'entrées/sorties. L'algorithme développé dans cette partie consiste à comparer les paramètres d'une demande du service avec les paramètres des services

disponibles. L'originalité de l'approche que nous proposons consiste donc à utiliser une sémantique précise. Les services sont annotés avec les types de données abstraits et les types de données sémantiques. Les types de données sont considérés comme des paramètres potentiels dans l'algorithme de découverte de services. Basant sur ces critères, des services pertinents assurant chaque opération de la requête seront sélectionnés afin d'être composés. Le client présente la suite d'opération qu'il désire exécuter, l'algorithme parcourt l'ensemble des services Web qui peuvent exécuter l'opération en cours, si un service existe, l'algorithme met à jour la liste des services découvert, puis il passe à l'opération suivante jusqu'à ce qu'il épuise la suite d'opérations présentée.

*** Boolean** MatchInputsOutputs (ReqInputList, ServInputList,
ReqOutputList, ServOutputList):

Entrée: La liste des paramètres in/out des opérations de la requête et des opérations de chaque service;

Sortie: Booléen qui indique l'existence ou pas des services satisfaisants toutes les opérations de la requête;

Boolean *Satisfait;*
Begin
Satisfait := ***true****;*
for each $oper_i \in OperReq$
$E_{InOut}[i] := \varnothing$;
for each $S_j \in S$
for each $oper_k \in OperServ$
if *(ReqInputList = ServInputList) and (ReqOutputList = ServOutputList)* ***then***
begin
Add IndiceService and OperServIndice to E_{InOut} ;
{$E_{InOut} = E_{InOut} U \{indiceS_j$ *and* $indiceOs_j$*, Ajouter* S_j *qui répond à* I_i *et* O_i *à la liste du tableau* $E_{InOut}[i].\}$
end;
end for
end for
if $E_{InOut}[i] = \varnothing$ ***then*** *Satisfait :=* ***false;***
end if
end for
return *satisfait****;***
End.

La requête ne peut être satisfaite {Une des entrées du tableau est vide}. Fin du parcours de la boucle des opérations c-à-d on ne vérifie pas l'opération suivante. {No services that match Input and Output of operation found};

```
* FindSWS (Boolean a);
Entrée (s) : liste de services découverts qui satisfont chacune des opérations.
Sortie (s) : service simple/composite qui satisfait la requête.
Boolean b;
 Begin
  b:= MatchInputsOutputs(ReqInputList,ServInputList,
                  ReqOutputLis,ServOutputList)
  if (b = true) then begin
    for each liste_i ∈ E_InOut do
     if ∩ (liste_i) = S_j then
      {Cette requête peut être satisfaite par un seul
         service S_j sans composition}
     else
      SWSComposition (Array P);
    end for;
 end if
End.
```

Cet algorithme consiste à sélectionner une combinaison de services pouvant participer dans la composition à partir des listes de services retournés par l'Algorithme 1 en cherchant à fournir en sortie une fonction composite entre chaque deux services. Cette phase détermine donc les fonctions nécessaires à la composition de services et ensuite propose un ordre d'exécution de ces fonctions (tâches). Cet ordre est transmis comme paramètre à la phase suivante de **FS4WSC** qui est la phase d'exécution de services, afin d'obtenir les valeurs pour les services candidats.

Avant de faire la composition de services, l'algorithme vérifie si les opérations de la requête s'exécutent séquentiellement et/ou parallèlement, en faisant la comparaison entre la liste des Outputs de la $i^{ème}$ opération ($Oper_i$OutputList) et la liste des Inputs de la $i^{ème}$ +1 opération ($Oper_{i+1}$InputList). Si les opérations s'exécutent en séquence, on fait la composition séquentielle de services correspondant à chacune de ces opérations, sinon on fait la composition parallèle.

```
 * SWSComposition (Array OperInOut)
Entrée: liste des opérations de la requête ;
Sortie: service(s) composé(s) qui satisfait la requête ;
begin
  for each Oper_i ∈ OperInOut do
    if Oper_iOutputList = Oper_{i+1}InputList then
     SéqCompo (S_{j/j∈{1..n}} ∈ E_inOut[i+1] , S_{j/j∈{1..m}} ∈ E_inOut[i])
    else
     ParCompo (S_{j/j∈{1..n}} ∈ E_inOut[i] , S_{j/j∈{1..m}} ∈ E_inOut[i+1]);
  end for
End.
```

3. Etude de cas : l'agence de voyages

3.1. Choix du langage fonctionnel CAML

Le but principal du choix du langage typé CAML [5] est d'apporter un moyen simple et naturel pour la définition de nouveaux types de données en privilégiant l'aspect sémantique des services Web (sémantique associée aux noms des services Web, aux opérations, aux entrées/sorties). L'utilisateur exprime, via ce langage, sa perception sémantique des différents objets qu'il souhaite implémenter et non pas les détails techniques de mis en œuvre.

Afin d'implémenter les annotations sémantiques associées aux services Web de notre exemple, nous avons intégré de nouveaux types de données sémantiques dans le langage CAML après avoir décrit nos services, on se basant sur les types de base de ce langage.

3.2. Exemple d'application

Une agence de voyages *"e-TravelAgency"* fournit typiquement les services pour: consultation, réservation, paiement et annulation de billets d'avion, de chambres d'hôtel et de locations de voiture. Afin de fournir ces services à ses clients, l'agence de voyages doit établir des liens avec d'autres entreprises : compagnies aériennes, compagnies de location de voitures, réseaux hôteliers et une institution financière (une banque).

Un enseignant chercheur, habite à "*Alger*", doit se rendre à "*Constantine*" *Samedi* pour assister à un colloque. Il décide d'organiser son voyage par Internet en faisant appel à différents services Web de l'agence "*e-TravelAgency*". Sa requête comprend la réservation de billet d'avion et la location d'un véhicule pour la durée du séjour, ainsi que le payement de ces derniers.

Afin de satisfaire la demande du client, on propose la composition des services web montrée dans la Figure 3.

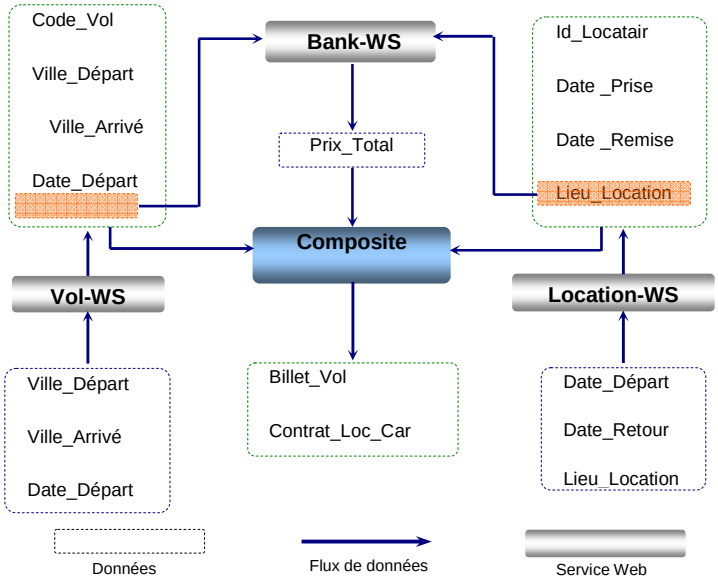

Figure 3 : Composition de Services Web Vol-WS, Location -WS et Bank-

3.3. Contraintes sémantiques

La représentation de date diffère d'un service à un autre. Le service Web de *"Location de voiture"* utilise un format (jj.mm.aaaa), alors que le service Web "*Rent-a-Car*" adopte une notation anglophone (mm.jj.aaaa). Cet exemple montre qu'une description des données établie par le langage WSDL ne remplit pas les exigences de l'échange sémantique. Afin de prendre en considération ces deux formats un type *date* est défini à l'aide des constructeurs *App* et *Fun* comme montré ci-dessous.

Le service "*Bank-SW*" possède une fonction *Payer* qui prend en entrée les paramètres "*prix*" correspondant aux différentes réservations : vol, hôtel, activité, location de voiture, et retourne en sortie le "*prix total*" des différentes réservations. Ce service utilise la devise "*Euro*" du prix total, tandis que le prix utilisé dans les services "*Reserv-Vol-SW*" et "*Rent-a-car-SW*" est le "*DA*". Malgré une compatibilité des types de données (type *"float"*) et des concepts sémantiques utilisés (concept *"prix"* et concept *"price"*), les données doivent être interprétées différemment, à cause de l'incompatibilité des unités de mesure. Pour éliminer cette incohérence, il faut convertir le prix des services à additionner dans la même devise du service "*Bank-SW* ". A cet effet, notre système offre des fonctions de conversion du ***DA*** vers l'***Euro***, du ***Yen*** vers l'***Euro*** et de l'***USD*** vers l'***Euro*** …etc. car la banque utilise seulement la monnaie qui est mesurée en *Euro* dans notre exemple. Ces fonctions sont définies comme suite : "*DA_To_Euro*", "*USD_To_Euro*" etc. Pour faire cette

conversion il suffit seulement de définir des formules qui permettent de multiplier par un facteur multiplicateur qui est un nombre utilisé pour la mise à l'échelle, selon la monnaie de chaque pays qui est définie dans l'ontologie.

```
# type monnaie = {nom : string; valeur : float};;

# let prix_vol = {nom= "euro"; valeur= 999.5};;

# type  date =

        | App of jour_semaine * int * nom_mois * int ;

        | Num of jour_semaine * int * int * int ;

        | Fun of nom_mois * jour_semaine* int;

# let dA_To_EURO x = x *. 0.01;;

val dA_To_EURO : float -> float = <fun>
```

Cependant, l'intérêt de la description sémantique est de spécifier les paramètres du service en leur donnant une signification.

Pour récapituler, le schéma suivant montre les étapes à suivre pour aboutir à une composition satisfaisant la requête.

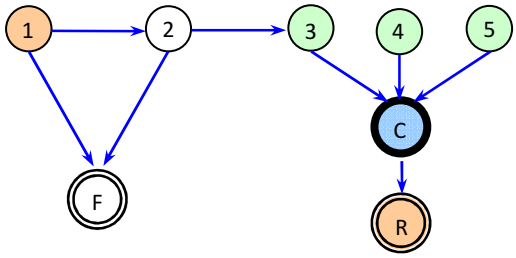

Figure 4 : Schéma de fonctionnement du système

(1). Matching entre requête et services: Appariement se fait entre le contexte et/ou les annotations de la requête et des services.

(2). Matching entre les opérations de la requête et les opérations de chaque service : Appariement se fait entre le contexte et/ou les annotations des opérations de la requête et des services.

(3). Matching entre les listes d'Inputs et les listes d'Outputs de la requête et des Services : Appariement se fait entre le contexte et/ou les annotations des paramètres d'Inputs/Outputs de la requête et des services satisfaisant chaque opération de la requête.

(4). Ajouter l'indice du service et de l'opération du même service à la liste des services satisfaisant l'opération de la requête.

(5). Matching entre les opérations de la requête : Appariement entre la liste des Outputs de la $i^{ème}$ opération et la liste des Inputs de la $i^{ème}$ +1 opération afin de déduire le type de la composition des services (séquentielle et/ou parallèle).

(C). Composition des services Web.

(R). Résultat de la composition (Requête satisfaite).

(F). Aucune correspondance trouvée / Fin du traitement (Requête non satisfaite).

Conclusion & Travaux futurs

Afin de permettre une composition dynamique, nous nous sommes basés sur l'*approche fonctionnelle*. Nous avons proposé *un système de programmation fonctionnelle (FS4WSC)* qui permet de *composer les services Web* avec l'étude de la description sémantique de ces services, où nous avons défini un formalisme qui permet de raisonner sur des structures de données, garantir la sémantique de la requête et des services Web et est muni des constructeurs pour composer les services. L'objectif recherché à travers l'utilisation de la sémantique est de permettre aux machines d'interpréter les données traitées et de saisir leurs significations de manière automatique. Cet objectif est concrètement atteint par l'annotation sémantique des services.

L'approche que nous avons proposée permet de rendre explicite la représentation des services Web et la sémantique qui leur sont associés, ainsi que la composition des services grâce à l'utilisation des *fonctions* et la *composition de fonctions.* Elle permet aussi à l'utilisateur de profiter de la simplicité et la puissance d'expression que les langages fonctionnels offrent et de la richesse de la sémantique claire de ces langages. Elle offre des possibilités très étendues de définition et de manipulation de structures de données très riches.

En conclusion, les travaux proches du notre se préoccupent principalement de la sémantique de services. Nous jugeons pertinent d'utiliser l'architecture classique des

services Web employée dans les organisations afin que la mise en œuvre de l'annotation sémantique ne modifie pas les systèmes existants.

De nombreuses perspectives peuvent être envisagées. Par exemple, l'introduction d'une sémantique formelle pour la vérification de la validité d'une composition. L'objectif d'une telle modélisation est de permettre l'utilisation de la puissance des langages formels pour effectuer des vérifications sur la validité d'une composition, vérifier des propriétés sur le service composite (e.g. l'absence d'interblocage) ou encore faciliter le processus de composition en permettant d'évaluer par exemple la composabilité de deux services, ou la remplaçabilité d'un service par un autre.

Un autre aspect peut également être traité, il s'agit de développer un algorithme d'appariement (matching) qui fait des correspondances entre les pré-conditions d'une action avec les effets d'une autre, ainsi que la prise en compte des besoins non-fonctionnels de l'utilisateur, comme par exemple des besoins de qualité de service et de sécurité.

Références

[1] Eijck, J.: Computational Semantics and Type Theory, Chapters 1-6. http://homepages.cwi.nl/~jve/cs/cs.pdf (2003)

[2] Preeda, R.: M.S Essay Computational Semantics and Type Theory and its influence on Semantic Web and Related Applications – An Introduction. (Under the Direction of John A. Miller) LSDIS Lab, Computer Science Department, University of Georgia, Athens, 30602 (2004)

[3] Bernard, D.: (Président du BIT Group) SOA : de l'entreprise-silo à l'entreprise-réseau. www.guideinformatique.com/lettrefiche-soa_de_lentreprise_ silo _a_lentreprise_reseau-437.htm, la lettre de mars (2007)

[4] Daniel, J.: SERVICES WEB Concepts, techniques et outils. Edition vuibert Informatique (2003)

[5] Maria-Virginia, A.: Introduction au noyau fonctionnel du langage Ocaml (2005 – 2006)

[6] Maesano, L., Bernard, C., Le Galles, X.: Services Web avec J2EE et .NET Conception et implémentations. Editeur Eyrolles (2003)

[7] Jean-Marie, C. : Services Web avec SOAP, WSDL, UDDI, ebXML... © Eyrolles (2002)

[8] Curbea, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N. and Weerawarana, S. Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. IEEE Internet Computing, 6(2). (2002)

[9] Dustdar, S., Schreiner, W.: A survey on web services composition. Int.J. Web and Grid Services, Vol. 1, No. 1, pp.1-30. (2005)

[10] Hadjila, F., Chikh, M.: La composition des services Web sémantiques : une approche à base de distance sémantique. Siie'2008 hammamet, Tunis. (2008)

[11] Hadjila, F., Chikh, M.: Une approche orientée agents pour la composition sémantique des

[12] services Web. Lania Chlef, Algérie. (2007)