Résumé

Le sujet de ce mémoire se situe au cœur de la problématique posée actuellement dans le contexte des services web et de l'Internet, l'automatisation de la découverte des services web basée sur la sémantique de leurs descriptions est un problème en cours de recherche.

Les services web sont des applications modulaires qui communiquent entre eux à travers le réseau Internet, autrement dit, ils représentent un mécanisme de communication entre applications distantes à travers le réseau Internet. Ils mettent en valeur la fonctionnalité du web par le changement de sa nature du document à orienté-service. Les services web peuvent être utilisés par les humains ou programmes afin d'accomplir une tache donnée. Pour bénéficier de leurs fonctionnalités, un mécanisme de découverte efficace pour la localisation et la sélection du service approprié est exigé.

Dans ce mémoire, nous avons proposé un algorithme de matching des spécifications des services web avec celle de la requête de l'utilisateur basé sur la substitution des contraintes. Cet algorithme se base sur le schéma SWSMF (*Semantic Web Services Matching Framework*). Une autre contribution réside dans la proposition d'une approche de composition dynamique où le processus du matching des services web s'effectue indépendamment les uns des autres, qui s'appuie sur les capacités des agents cognitifs, qui sont trois types : l'agent utilisateur, l'agent médiateur et les agents fournisseur. Un protocole de communication entre ces agents est décrit. Cette approche offre une flexibilité en matière d'ajout, de suppression et de modification des services web.

Mots clé :

Services web, web sémantique, ontologies, services web sémantiques, composition des services, matching.

Abstract

The topic of this memory is located at the heart of the problematic posed currently in the context of web services and Internet. The semantic-based web services discovery is a problem in the research worry.

Web services are modular applications that communicate with one another over the Internet network, in other words, they represent a communication mechanism between distant applications over the Internet network. They enhance web functionality by altering its nature from document to service oriented. Web services can be used by humans or programs in order to accomplish a particular task. To benefit from their functionalities, an efficient discovery mechanism for locating and selecting the appropriate web services is required.

In this memory, we have proposed a matching algorithm of the web services specifications with the ones of the user request'. This algorithm is based on the SWSMF scheme (Semantic Web Services Matching Framework). Other contribution consists in the proposition of a dynamic and a composition approach where the matching process for web services is doing independently, which use the cognitive agent capacities, which are three; user, mediator and provider agent. A communication protocol between those agents is described. This approach offers flexibility in the addition, removal and modification of the web service.

Keywords:

Web services, web semantic, ontology, semantic web services, service composition, matching.

Remerciements

Premièrement, je remercie le Dieu de m'avoir donné le courage et la patience pour accomplir ce travail.

Je tiens à exprimer ma gratitude et mes vifs remerciements à Monsieur **TARI Zahir,** professeur à l'institut RMIT, Melbourne, Australie, d'avoir accepté d'être mon encadreur durant cette année de magistère et pour la confiance qu'il m'a donnée et ses précieux conseils ;

Je tiens à remercier vivement Monsieur **TARI Abdelkamel,** docteur à l'université de Béjaia, responsable de l'école doctorale d'informatique et chef de département d'informatique de Béjaia, pour son suivi minutieux, ses conseils judicieux, son aide précieuse ainsi le temps qu'il a consacré au bon déroulement de ce travail.

Je remercie également le président de jury pour m'avoir fait honneur de juger mon travail. Je remercie aussi les membres de jury pour les critiques et les remarques portées à mon rapport.

Que tous les étudiants de l'école doctorale ReSyD4 trouvent mes remerciements pour l'environnement et l'ambiance du travail que nous avons partagés durant ces deux années de magistère.

Enfin, je remercie tous les Professeurs qui ont assuré ma formation du magistère.

Dédicaces

Avec un grand respect et une profonde gratitude, je dédie ce travail aux êtres les plus chers, ceux qui voulaient ma réussite.

A la mémoire de mon père, qui m’a donné tout ce qu’il possède afin que rien ne m’empêche de réussir. Que Dieu t’accueille dans son vaste paradis.

A ma très chère mère qui s’inquiète toujours sur moi. Que Dieu te garde pour nous.

A la mémoire de mes grands-mères. Que Dieu vous accueillent dans son vaste paradis.

A mes très chers frères et sœurs.

A ma très chère Naïma qui m’a soutenu énormément.

A Achour, Kamel, Salah, Rabah, Redouane et Souhila.

Sommaire

CHAPITRE II
La découverte et la composition des services web

CHAPITRE III
Etat de l'art sur le service du matching

Liste des figures

Liste des tableaux

Liste des algorithmes

Liste des acronymes

abréviation	signification
API	Application Programming Interface
BPEL4WS	Business Process Execution Language for Web Services
CBSDP	Context Based Service Discovery Protocol
Compaq	COMPatibility And Quality
CPP	Collaboration Protocol Profile
CPA	Collaboration Protocol Agreement
CORBA	Common Object Request Broker Architecture
DAML-S	DARPA Agent Markup Language Service
DARPA	Defense Advanced Research Projects Agency.
EAI	Enterprise Application Integration
EDI	Electronic Data Interchange.
EbXML	Electronic Business using eXtensible Markup Language
FTP	File Transfer Protocol
HTTP	HyperText Transfer Protocol
HP	Hewlett Packard
IBM	International Business Machines
Intel	Integrated Electronics
KIF	Knowledge Interchange Format
MIME	Multi-purpose Internet Mail Extension
OASIS	Advancing Open Standards for the Information Society
OWL	Ontology Web Language
OWL-S	Ontologie Web Language-Service
OML	Ontology Markup Language
PSM	Problem-Solving Method
RDF	Resource Description Framework
RDF-S	Ressources Description Framework - Sheme
SMTP	Simple Mail Transfer Protocol
SOA	Services Oriented Architecture
SOAP	Simple Object Access Protocol
SUN	Stanford University Network
TCP/IP	Transmission Control Protocol / Internet Protocol
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier.
URL	Uniform Resource Locator
UPML	Problem-solving Method Language
UTF8	Unicode Transformation Format 8 bits
WSDL	Web Service Description Language
WSDA	Web Service Discovery Architecture
WSFL	Web Services Flow Language
WSOA	Web Services Oriented Architecture.
XMLP	XML Protocol
XDD	XML Declarative Description
XOL	XML-based Onrology Exchange
XML	eXtensible Markup Language
XSD	XML schéma Datatypes
XSLT	XSL Transformation language
XL	eXpérimental Language
WfMC	Workflow Management Coalition

INTRODUCTION GENERALE

Introduction Générale

1. Généralités

L'émergence d'Internet est entrain de révolutionner la manière dont les entreprises interagissent avec leurs partenaires. Durant ces dernières années, le nombre et la variété des ressources disponibles sur Internet ont augmenté considérablement conduisant à de nouvelles formes d'interactions telles que le B2B et le B2C.

Avec le développement rapide d'Internet, il devient de plus en plus intéressant pour les entreprises d'utiliser un support électronique pour leurs transactions commerciales. Le commerce électronique peut être défini comme étant l'utilisation des télécommunications et des systèmes d'information pour effectuer des transactions commerciales entre entreprises et individus. Plusieurs problèmes restent à résoudre afin de faciliter l'utilisation de l'Internet et de profiter pleinement des bénéfices du commerce électronique.

SOC (*Service Oriented-Computing*) permet aux technologies de développer et de connecter (collaborer) différentes applications dans différents domaines, comme le e-commerce, e-science …, pour cela SOC utilise la publication des services, comme éléments fondamentaux pour cette tâche. Le problème majeur est comment localiser ces services avec quelques informations de leurs existences. Ce problème est nommé le problème de découverte des services web.

Les services web sont un paradigme naissant qui vise à la transposition des architectures par composant dans le cadre du web. Il s'agit d'une technologie jeune ; née à la fin des années quatre-vingt-dix, par quelques entreprises comme Microsoft, IBM et Sun. Ils prennent leur origine dans l'informatique distribuée et dans l'avènement du web. Le but de l'informatique distribuée est de permettre à une application sur une machine et d'accéder à une fonction d'une autre application sur une machine distante et ce, de la même manière que l'appel d'une fonction locale, indépendamment des plates formes et des langages utilises.

L’objectif de l’approche des services web est de transformer le web en un environnement distribué de calcul où les programmes (services), qui peuvent interagir de manière intelligente en étant capables de se découvrir automatiquement, de négocier entre eux et de se composer en des services plus complexes. En d’autres termes, l’objectif c’est de mieux exploiter les technologies de l’Internet en substituant, autant que possible, les humains, par des machines afin de permettre une découverte et une composition automatique des services web. Un autre objectif des services web est de faciliter l'accès aux applications entre entreprises et ainsi de simplifier les échanges de données.

Le problème de découverte de service est la localisation des services publiés sans la connaissance a priori de leur existence [59], il se situe dans la comparaison de la description des services avec celle de la requête des utilisateurs afin de trouver le ou l’ensemble des services qui satisfont ces requêtes. Il y a différentes approches de découverte de services. Une partie du problème de la découverte de service est le problème du matching.

Les approches existantes du matching nécessitent l’intervention de l’être humain afin de déterminer le résultat correct, ce qui veut dire que le choix du résultat est fait manuellement, ce qui est l’obstacle majeur pour automatiser le processus de découverte et de composition de service. Pour surmonter ce problème, le *matchmaker* doit être capable de déterminer automatiquement la pertinence de résultat du *matching.*

Le *matchmaker* est un programme désigné pour accepter les requêtes et de donner une réponse pour ces requêtes en appliquant une technique de matching donnée [59]. Afin d’aboutir à un matching correct, le *matchmaker* doit extraire les spécifications fonctionnelles de haut niveau du service et de la requête de l’utilisateur, ces spécification concernent leur but, leurs rôles, leurs contextes et leurs comportement extérieurs. Il doit aussi extraire la sémantique du domaine d’application afin qu’il puisse faire la médiation entre les requêtes et les services.

2. La problématique du sujet

Les services web sont désignés pour fournir une interopérabilité entre les applications métiers, cette technologie présume une forte interaction. L’aspect que nous avons traité dans notre travail est la découverte sémantique de ces services, plus précisément le problème du matching. Dans ce sujet, nous ne nous sommes pas intéressés au problème de l’invocation et de l’interblocage durant l’exécution. Le matching est un aspect très important dans les interactions de E-commerce. La découverte concerne la localisation des services publiés sans avoir la connaissance sur leur existence. Dans le processus de la découverte, nous rencontrons le problème du matching, où les descriptions des services sont comparées aux exigences de la requête afin de trouver une correspondance et retourner les services appropriés à la requête. L’automatisation de la découverte fournit beaucoup d’avantage. Plusieurs approches dans la littérature ne sont pas adoptées pour automatiser le processus du service matching car elles ne prennent pas en compte la réalisation des objectifs de l’utilisateur [59] ce qui nécessite l’intervention humaine afin de choisir le bon résultat qui satisfait son but.

Les spécifications des services web peuvent être classées en plusieurs types, on retrouve les spécifications fonctionnelles et non fonctionnelles, on retrouve aussi dans le premier type des spécifications celles de haut niveau comme l’objectif et le comportement de réalisation de but, et celles de bas niveau comme les interfaces d’accès et les protocoles utilisés. Pour les spécifications non fonctionnelles, deux catégories existent : des spécifications techniques

comme le prix et les spécifications non techniques comme les exigences de la qualité de service. Pour avoir un matching exacte, le processus du matching doit être fait dans chaque niveau, il doit s'appuyer sur les spécifications fonctionnelles de haut niveau, car à ce niveau de spécifications est extraite la sémantique du service. Les autres spécifications servent comme filtre pour les résultats retournés par la première étape du matching.

Le souci est comment réaliser un matching correct, qui prend en compte la sémantique des services et de la requête. La réalisation de l'algorithme du matching doit se baser sur la représentation de ces spécifications. Les approches qui considèrent la représentation de la requête et du service sous forme de conditions d'entrée et de conditions de sortie et voient le processus de réalisation du service comme une boite noire, manquent de la sémantique et ne prennent pas en compte le comportement interne. Pour cela, la question qui se pose, c'est comment trouver une représentation des spécifications de telle sorte la représentation de la sémantique est incluse.

On retrouve dans les approches qui considèrent le comportement des services comme un ensemble d'états, les approches one-to-one, qui sont rigides, et les approches m-to-n qui sont plus adaptées dans le cas ou la séquence d'états de la requête n'est pas égale à celle du service. Ces dernières approches sont généralement lourdes, car la complexité de ses algorithmes est exponentielle. Alors réaliser un algorithme qui accepte deux séquences d'états de tailles différentes est très important.

Le résultat retourné pour l'utilisateur peut être un seul service ou un ensemble de services. Le matching dans cette situation s'effectue pour un ensemble de services. Dans ce cas du matching, la complexité reste très importante, voir impossible dans des cas ou on doit prendre en compte la complexité en temps d'exécution (le temps de réponse). La réalisation d'une bonne approche de composition qui prend en charge la sémantique décrite auparavant est aussi l'axe de notre recherche.

3. Les limitations des travaux existants

Différentes approches ont été proposées dans la littérature pour réaliser la découverte dynamique de services. Toutes ces approches implantent en fait une découverte approximative car il n'est pas réaliste d'imaginer qu'il y a toujours un service qui correspond exactement aux besoins spécifiés. Ces approches diffèrent par le langage de description de services utilisé (*e.g.*, DAML-S, logique de description) ou par *l'algorithme de découverte* utilisé (*matchmaking* [49], *test de subsumption, réécriture* [6]). Par exemple, [11] propose d'utiliser des ontologies de processus pour décrire le comportement des services et définit un langage d'interrogation de processus (*Process Query Language*) pour interroger ces ontologies. [18] définit une ontologie basée sur le langage DAML pour décrire des ressources mobiles et proposent un processus de correspondance qui localise les ressources en fonction de leurs caractéristiques. Le processus de correspondance s'appuie sur des règles qui exploitent l'ontologie, les profils des services et la requête du client pour réaliser une correspondance à partir des relations attributs-valeurs. D'autres approches basées sur une description DAML-OIL des services proposent d'exploiter les mécanismes de raisonnement fournis par DAML-OIL pour supporter la découverte dynamique des services web. [64] propose un algorithme de correspondance plus élaboré entre des services et des requêtes décrits en DAML-S. L'algorithme reconnaît différents degrés de correspondance qui sont déterminés par la distance minimale entre les concepts dans la taxonomie de concepts. De la

même façon, le système ATLAS [50] opère sur des ontologies DAML-S et utilise deux ensembles séparés de filtres : (i) les attributs fonctionnels et (ii) les E/S des services.

Les approches du matching direct (*services direct matching*) sont classées en deux catégories [59]: les approches structurées et les approches non structurées. Les approches structurées peuvent différencier entre les spécifications du service web, tandis que les approches non structurées utilisent les techniques *keyword-based matching* dans le processus du matching et cela du fait qu'elles ne différencient pas entre les spécifications de service. Les approches structurées peuvent être fonctionnelles ou non fonctionnelles.
Les services de matching existants comme ceux qui sont décrits dans [13] [64] traitent les services comme étant des documents et font de telle sorte à maximiser le nombre de services retournés et laisse à l'utilisateur le choix du bon service. Les différent concepts obtenus c'est en examinant les différentes descriptions, ces concepts seront matchés utilisant des approches syntaxiques [58] [54] ou par des approches sémantique moins efficaces [17].
Le matching des concepts obtenus utilisant une approche syntaxique telle que [53] est imprécise. Elle oblige que les requêtes et les descriptions des services doivent être du même vocabulaire. Le matching utilisant des approches sémantiques prend en considération le besoin de médiation entre les concepts. La caractéristique commune entres ces approches c'est que le matching utilise le concept de *subsumption rules* [52] ou en utilisant une forme logique [20].
Utilsant *aggregate service matching approches* exige que les descriptions de services et les requêtes des utilisateurs ayant le même vocabulaire. Ils sont rigides de telle sorte ils n'adaptent aucune technique de médiation entre les déscriptions de services et des requêtes des utilisateurs [59].

4. Objectifs et composition du mémoire

L'objectif de notre travail est d'étudier le problème de l'automatisation de la découverte des services web en se basant sur leurs sémantiques et de venir par un algorithme du matching des services web. Notre algorithme du *matching* des spécifications fonctionnelles par substitution de contraintes repose sur FSMS (*The Functional Substitutability Matching Scheme*), un schéma du matching proposé par Tari et Elgadawi [59,22] qui utilise les fonctionnalités de services comme aspect de comparaison et la substitution comme principe de matching, notre travail constitue une extension des travaux de Tari [59] et les principes qu'il a utilisés dans l'approche proposée. Dans une première partie, nous essayons de réaliser un algorithme de matching direct des services, de type m-to-n, nous avons utilisé le principe de substitution de contraintes dans la vérification de la correspondance entre les spécifications de la requête et celles du service web. Par la suite, la conception d'une approche de composition qui utilise l'algorithme du matching réalisé est notre objectif. Pour cela nous l'avons partagé en cinq chapitres essentiels.

Dans le premier chapitre, nous avons donné les concepts fondamentaux sur les services web, il est constitué de trois sections : dans la première section, nous avons introduit les principes et les composants de l'architecture SOA, l'architecture et les technologies des services web. Dans la deuxième section, des notions sur web sémantique et les ontologies sont introduites. Dans la troisième section, nous retrouvons les principes et la modélisation des services web sémantique.

Nous résumons, dans le deuxième chapitre, quelques problèmes reliés à la découverte des services web en citant différentes approches existantes de la découverte de service. Nous essayons de faire dans la deuxième partie de ce chapitre, un aperçu sur la composition des services web ainsi de présenter les techniques suivies dans cette direction de recherche.

Le troisième chapitre, nous le consacrons pour un état de l'art sur l'axe de notre travail, nous commençons par un état de l'art sur la présentation et la découverte de service, nous poursuivons par les détails des techniques et du schéma sur lesquels se base notre proposition et que nous voulons améliorer, notre idée a été inspirée du travail qui a été fait par A. Tari et I. Elgedawy [59, 22]. Nous finissons ce chapitre par la présentation de quelques travaux reliés au matching direct et par agrégation des services web.

Notre contribution vient dans le quatrième chapitre. Dans la première section, nous avons donné quelques définitions et concepts utilisés dans notre algorithme. Ensuite, dans une autre section, nous avons proposé un algorithme de matching des spécifications fonctionnelles de haut niveau, ainsi quelques concepts et théorèmes sur lesquels est basé notre algorithme.

Dans le dernier chapitre, nous avons proposé une approche dynamique de la composition des services web. Cette approche repose sur les capacités des agents cognitifs pour former un système distribué qui se base sur un protocole de communication entre les agents décrit dans la même section. Nous avons utilisé trois types d'agent, un agent *utilisateur* qui initie la composition en envoyant une requête à l'agent *médiateur*, ce dernier redistribue la requête aux agents *service* qui ont le rôle d'effectuer le matching. Lorsque le matching est terminé, ils envoient les résultats à l'agent médiateur qui s'occupe de trouver le plan de composition approprié.
Nous terminons notre mémoire par une conclusion et des perspectives.

CHAPITRE I

Concepts fondamentaux sur les services web

Chapitre I

Concepts fondamentaux sur les services web

I.1. Les services web

I.1.1. Introduction

De plus en plus avec l'essor d'Internet, le développement tend vers les technologies du web. Il est difficile de faire la distinction entre les différents logiciels qui sont de plus en plus intégrés au web. Dans ce contexte, la notion de services électroniques apparaît comme un nouveau paradigme prééminent pour l'informatique répartie et le e-business, et permettant de franchir une étape importante vers la construction du web sémantique.
Avec l'interconnexion des ordinateurs en réseau et en particulier à travers Internet, il devient possible de faire fonctionner des applications sur des machines distantes. L'intérêt d'une application fonctionnant à distance peut répondre aux problématiques suivantes :

- Les données peuvent être présentes uniquement sur le serveur distant.
- Exploitation de la puissance de calcul et les capacités de stockage du serveur distant et dont l'utilisateur local ne dispose pas.
- L'application distante peut être utilisée simultanément par un grand nombre d'utilisateurs et sa mise à jour n'intervient qu'à un seul endroit.

Pour toutes ces raisons, une interaction entre des programmes distants peut être utile. Les services web apportent une solution à ce problème en définissant d'une manière standard l'invocation d'une application distante et la récupération des résultats à travers le web.

I.1.2. L'évolution du web

Les services web sont considérés comme étant l'évolution du web. On peut distinguer trois phases de développement dans l'histoire du web :

- **Le Web du Document**, le phénomène Internet originel, utilisé principalement par des organisations et des particuliers pour publier des informations sur leur travail, etc.
- **Le Web Applicatif**, le progrès grâce auquel les entreprises ont commencé à utiliser le Web à des fins commerciales. Les applications qui gèrent les sites Internet reposent

sur des standards tels que les CGI (*Common Gateway Interface*) et les EJB (*Entreprise Java Beans*).

- **Le Web des Services** est la phase émergente, dans laquelle les serveurs d'application précédents communiquent entre eux. L'échange de données informatisées entre deux applications nécessite une normalisation des messages échangés.

Les services web constituent une avancée majeure dans le domaine de la coopération entre entreprises, c'est-a-dire les échanges B2B. World Wide Web (WWW), qui est devenu une infrastructure importante pour l'échange des informations. Pour la raison de son accessibilité facile et large, WWW est fait seulement pour l'être humain pour l'accès aux informations et la publication des services. Pour dépasser les barrières de cette technologie, les deux approches, le web sémantique et les services web, sont proposées.
Le web sémantique concerne à ajouter une information sémantique aux descriptions afin de rendre l'ordinateur capable de comprendre l'information et l'analyser sans l'intervention de l'homme. Les services web permettent l'intégration de plusieurs solutions et systèmes.
En combinant le web sémantique et les services web, on obtient des services web intelligents [24], qui permettent automatiquement l'interaction et la découverte des services sans l'intervention de l'homme. Si on classe l'évolution du web selon deux axes (le dynamisme et l'intelligence), on obtient le schéma suivant :

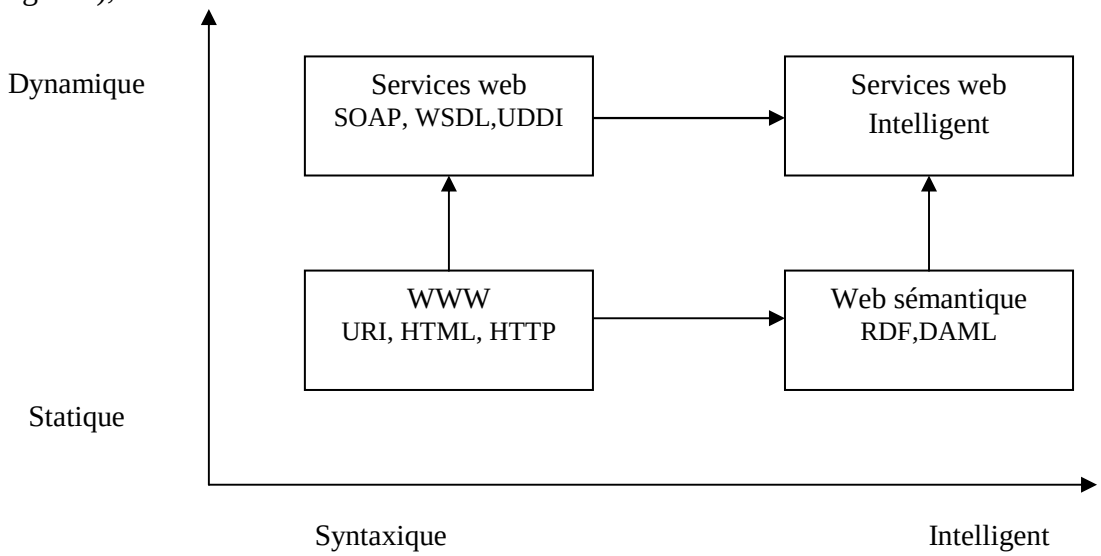


Figure I. 1. L'évolution du web

I.1.3. Définition des services web

Un service web est un composant logiciel représentant une fonction applicative (ou un service applicatif). Il peut être accessible depuis une autre application (un client, un serveur ou un autre service web) à travers les réseaux Internet en utilisant les protocoles de transports disponibles. Autrement dit, il représente un mécanisme de communication entre applications distantes à travers le réseau *Internet* indépendant de tout *langage de programmation* et de toute plateforme d'exécution en utilisant le protocole *HTTP* comme moyen de transport, en se basant sur les mécanismes d'appel et de réponse et en employant une syntaxe basée sur la notation *XML* pour décrire les appels de fonctions distantes et les données échangées.
Le consortium W3C[1] définit un service web comme étant :
« Une application ou un composant logiciel qui vérifie les propriétés suivantes :

[1] *http://www.w3.org/2002/ws/*

- *Il est identifié par un URI[1];*
- *ses interfaces et ses liens (binding) peuvent être décrits en XML[2] ;*
- *sa définition peut être découverte (dynamiquement) par d'autres systèmes logiciels ou services web ;*
- *Il peut interagir directement avec d'autres services web d'une façon décrite par sa définition, à travers le langage XML et en utilisant des protocoles Internet. »*

IBM définit les services web comme suit :
« Les services web sont la nouvelle vague des applications web. Ce sont des applications modulaires, auto-contenues et auto-descriptives qui peuvent être publiées, localisées et invoquées depuis le web. Les services web effectuent des actions allant de simples requêtes à des processus métiers complexes. Une fois qu'un service est déployé, d'autres applications (y compris des services web) peuvent le découvrir et l'invoquer ».

I.1.4. Les éléments de définition

Plusieurs acteurs définissent les services web par des caractéristiques technologiques distinctives. Certains éléments principaux nous apparaissent importants :

• *Une application logicielle identifiée par un URI (Uniform Resource Identifier).*

Un *service web* est une application autosuffisante en ce sens qu'elle effectue une seule tâche et que ses composantes décrivent ses propres entrées et sorties de telle sorte que d'autres logiciels qui invoquent le *service web* puissent interpréter ce qu'il fait, comment invoquer sa fonctionnalité et à quel résultat cet autre service peut s'attendre. Un URI est la façon d'identifier un point de contenu sur le Web. L'URI le plus connue est l'adresse d'une page Web, par exemple : http://www.univ-bejaia.dz.

• *Capacité des interfaces et liaisons (bindings) d'être publiées, localisées et invoquées via XML*. Un *service web* peut-être publié dans un registre situé à l'intérieur ou à l'extérieur d'un SI (*Système d'information*). Il peut être localisé en interrogeant le registre qui l'héberge. Une fois localisé, un *service web* peut être invoqué en envoyant une requête appropriée.

• *Capacité d'interagir avec d'autres composantes logicielles via des éléments XML et utilisant des protocoles de l'Internet*

L'une des bases des *services web* est l'utilisation de protocoles standards de l'Internet tels que HTTP (*Hypertext Transfer Protocol*, le protocole du Web), SMTP (*Simple Mail Transfer Protocol*, le protocole du courriel électronique) et XML. Contrairement à une page web ou à une application de bureautique, les *services web* ne sont pas destinés à une interaction humaine directe. Ils sont plutôt conçus pour être utilisés par d'autres logiciels.

• *Composante logicielle légèrement couplée à interaction dynamique*

On veut dire par légèrement couplée, que le *service web* et le programme (le consommateur de *service web*) qui l'invoque peuvent être modifiés indépendamment l'un de l'autre. Cela offre une flexibilité qui permet aux entreprises d'éviter les coûts engendrés par l'intégration via des communications fortement couplées. L'interaction dynamique signifie que le consommateur de *services web* peut localiser et invoquer ce dernier au moment de l'exécution du programme sans avoir à programmer cette habileté à l'avance.

Tous ces éléments permettent de mieux comprendre ce que sont les *services web*. Chacun de ces éléments exprime une facette de ce qu'il est maintenant convenu d'appeler *services web*. Une définition globale d'un *service web* est donc :

Une application logicielle, légèrement couplée, à interaction dynamique, identifiée par un URI, pouvant interagir avec d'autres composantes logicielles et dont les interfaces et liaisons

[1] *Unique Resource Identifier*
[2] *Extensible Markup Language*

ont la capacité d'être publiées, localisées et invoquées via XML et l'utilisation des protocoles Internet communs. Ils sont les bases permettant de construire des systèmes distribués et ouverts sur Internet, utilisant des technologies indépendantes des plates-formes.

I.1.5. L'architecture SOA

Une *architecture orientée services* (notée *SOA* pour *Services Oriented Architecture*) est une architecture logicielle s'appuyant sur un ensemble de services simples. L'objectif d'une architecture orientée services est de décomposer une fonctionnalité en un ensemble de fonctions basiques, appelées services, fournies par des composants et de décrire l'interaction entre ces services

I.1.5.1. Principes généraux de l'architecture SOA

Une architecture orientée services repose sur quelques principes :

- La notion de **service**, c'est-à-dire une fonction encapsulée dans un composant que l'on peut interroger à l'aide d'une requête composée d'un ou plusieurs paramètres et fournissant une ou plusieurs réponses.
- La **description du service**, consistant à décrire les paramètres d'entrée du service et le format et le type des données retournées.
- La **publication** (*advertising*). La publication consiste à publier dans un registre (*registry* ou *repository*) les services disponibles aux utilisateurs,
- La **découverte** (*discovery*) des services, qui recouvre la possibilité de rechercher un service parmi ceux qui ont été publiés,
- L'**invocation**, représentant la connexion et l'interaction du client avec le service.

I.1.5.2. Composants d'une SOA

Une architecture orientée service est constituée de trois composants :

Prestataire de service*:* C'est la source de la fonctionnalité des services. Il publie un contrat d'interface utilisé par les demandeurs pour accéder au service. Le contrat définit ce que fait le service et comment y accéder.

Agence de services (Annuaire): C'est un enregistrement des services disponibles. Chaque fournisseur publie son contrat d'interface à l'agence avec les informations à utiliser pour localiser le service.

Demandeur de service : C'est le client qui consomme le service, il utilise l'agence pour découvrir quels services sont disponibles, une fois un service est localisé le demandeur extrait le contrat d'interface correspondant de l'agence. Le client utilise le contrat d'interface pour comprendre comment (les méthodes) et où (les adresses) accéder au service.

I.1.5.3. Avantages et objectifs d'une architecture SOA

Une architecture orientée services permet d'obtenir tous les avantages d'une architecture client-serveur et notamment :

- Une modularité permettant de remplacer facilement un composant par un autre.
- Une réutilisabilité possible des composants.
- De meilleures possibilités d'évolution et une maintenance facilitée.
- La distribution pour pouvoir utiliser ces composants à distance.
- L'interopérabilité pour l'utilisation des modules sur plusieurs plates-formes.
- Une plus grande tolérance aux pannes qui n'influe pas sur les autres services.

I.1.6. L'architecture des services web

L'architecture d'un service web se base complètement sur l'architecture SOA. Elle reprend, les mêmes acteurs qu'une architecture SOA,

I.1.6.1. L'architecture de référence (le modèle d'interaction)

Les interactions de base entre ces trois rôles incluent les opérations de publication (*advertising*), de recherche (*discovery*) et de liens (*bind*) d'opérations. Nous décrivons ci-dessous un scénario d'utilisation de cette architecture [43].

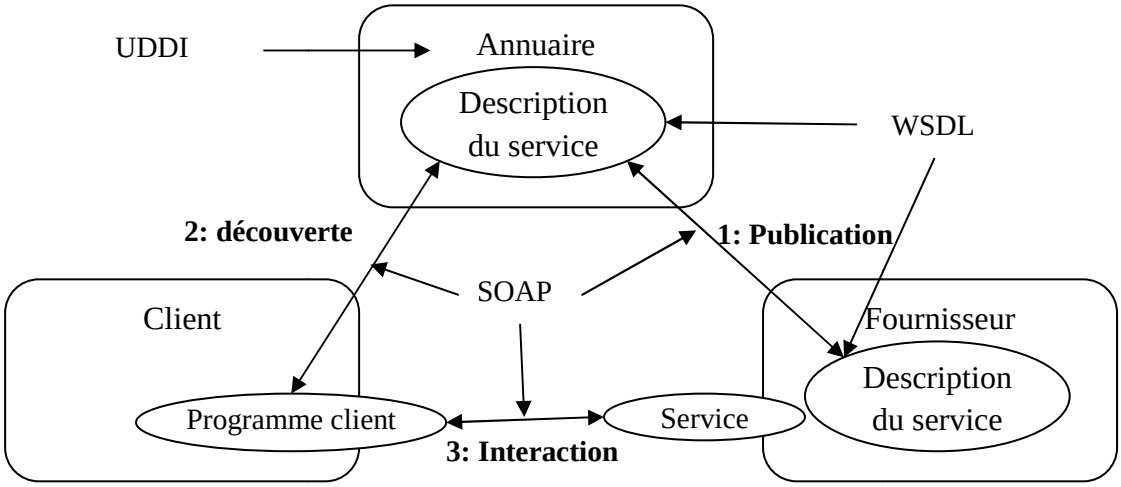


Figure I. 2. Modèle d'interaction des services web

1. Le fournisseur de services définit la description de son service et la publie dans un annuaire de service.

2. Le client utilise les facilités de recherche disponibles au niveau de l'annuaire pour retrouver et sélectionner un service donné.

3. Le client examine ensuite la description du service sélectionné pour récupérer les informations nécessaires lui permettant de se connecter au fournisseur du service et d'interagir avec l'implémentation du service considéré.

Pour garantir l'interopérabilité des trois opérations précédentes (publication, recherche et lien), des propositions de standards ont été élaborées pour chaque type d'interactions. Nous citons notamment quelques standards émergents :

- WSDL (*Web Services Description Language*): WSDL est un langage permettant de décrire les services web,
- UDDI (*Universal Description, Discovery and Integration*) fournit l'infrastructure de base pour la publication et les références des services web.
- SOAP (*Simple Object Access Protocol*): il définit la structure des messages échangés par les applications via Internet.

I.1.6.2. L'architecture étendue

Une architecture étendue est constituée de plusieurs couches se superposant les unes sur les autres, d'où le nom de pile des services web. La figure suivante décrit un exemple d'une telle pile [35].

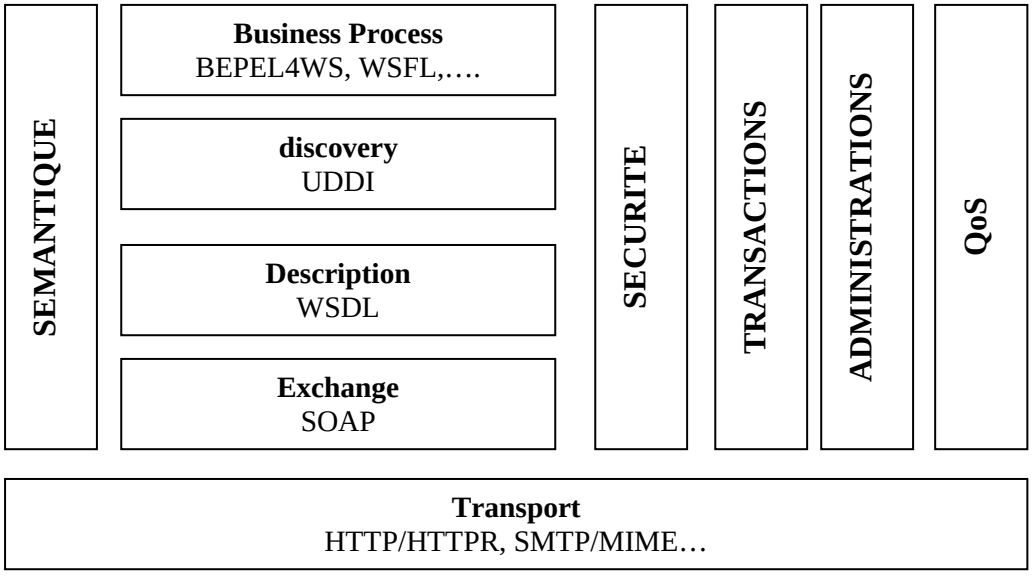


Figure I. 3. Architecture en pile (architecture étendue)

Nous mettons en relief trois types de couches distinctes ci-dessous :

- **L'infrastructure de base (Discovery, Description, Exchange) :** Ceux sont les fondements techniques établis par l'architecture de référence. Nous distinguons alors les échanges de message (établis principalement par SOAP), la description de service (WSDL) et la recherche de services que les organisations souhaitent utiliser (via le registre UDDI).
- **La couche Processus Métier (Business Process):** Cette couche supérieure permet l'intégration (composition) de services web. Elle établit la représentation d'un *Business Process* comme un ensemble de services web.
- **Les couches transversales (Administration, Transactions, Security, QoS) :** Ce sont ces couches qui supportent les différents standards mis à jour fréquemment, intervenant dans l'ensemble du processus d'un service web. Parmi ces derniers, nous pouvons mettre en relief la sécurité, l'administration, la qualité de service. Des travaux tentant d'intégrer le web sémantique dans ces couches transversales en ajoutant une couche verticale représentant le web sémantique et étant utilisable par les quatre couches horizontales représentant les standards [40].

I.1.7. Les technologies des services web

Il existe diverses technologies des services web, nous allons décrire les plus pertinentes en se basant sur leur architecture. Toutes ces technologies sont basées sur les méta-longages[1] XML (*eXtensible Markup Language*).

I.1.7.1. XML, une solution d'interopérabilité intéressante

L'interopérabilité implique que les spécifications du web (langages et protocoles) doivent permettre de garantir que les éléments matériels et logiciels soient compatibles entre eux. En clair, cela veut dire par exemple, deux programmes (exemple, un client et un serveur) sont interopérables s'ils peuvent échanger et interpréter des données correctement lors de leur première rencontre.

XML (*eXtensible Markup Language*) est une solution d'interopérabilité intéressante. Le langage peut être utilisé pour créer des messages destinés aux échanges entre des applications. Des applications qui utilisent ces messages comprennent la structure et la sémantique des

[1]Un méta-langage permet la spécification de langage par des règles.

données qu'elles reçoivent, ce qui était impossible avec HTML (*Hyper Text Markup Language*). XML permet à un client d'échanger des données avec beaucoup de sources de données ou de serveurs hétérogènes, sans connaître à l'avance les structures, les types et les divers formats des données qu'il reçoit. Cette transparence est précieuse pour la coopération.

I.1.7.2. La communication (échange) de données avec SOAP

SOAP **(***Simple Object Access Protocol*) est un standard du *consortium W3C* définissant un protocole qui assure des appels de procédures à distance RPC (*Remote Procedure Call*) s'appuyant principalement sur le protocole HTTP (*hyperText Transfer Protocol*) et sur XML, mais aussi sur SMTP (*Simple Mail Transfer Protocol*) et POP (*Post Office Protocol*). Il assure l'interaction entre les services web en transportant les paquets de données encapsulés sous forme de texte structuré.
Le principal objectif du protocole SOAP est de permettre la normalisation des échanges de données.
SOAP assure l'interopérabilité entre composants tout en restant indépendant des plates formes et des langages de programmation. Il repose sur les deux standards XML et HTTP respectivement pour la structure des messages et pour le transport. SOAP permet une interopérabilité avec divers environnements logiciels (Java/RMI, Corba (*Common Object Request Broker Architecture*), COM/DCOM, etc.) quelle que soit leur plate-forme d'exécution. Les paquets de données circulent sous forme de texte structuré au format XML.

I.1.7.2.1. Architecture du protocole SOAP

Le standard SOAP comporte trois parties principales :

- **Un framework de messagerie :** Le framework de messagerie SOAP requiert que les messages SOAP soient composés d'une enveloppe qui contient un Header et un Body.
- **un standard d'encodage :** Les règles d'encodage SOAP expriment les types de données définies pour l'application en XML.
- **un mécanisme RPC :** La spécification SOAP RPC décrit la possibilité d'utiliser le protocole SOAP pour invoquer les procédures sur réception de la fin du message SOAP.

I.1.7.2.2 Structure d'un message SOAP

La structure d'un message SOAP est constituée d'un entête HTTP, une enveloppe, qui contient un entête et un corps :

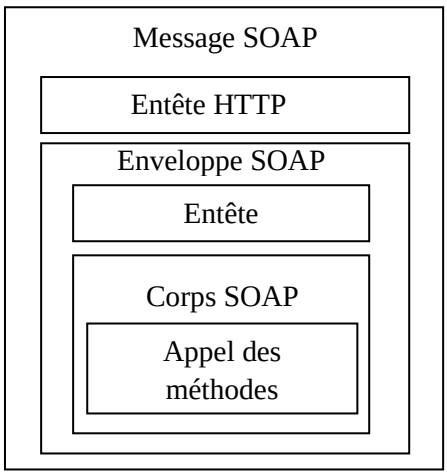


Figure I. 4. Format d'un message SOAP

- **Entête HTTP:** cet entête contient :
  - La version de HTTP utilisée.
  - La date de génération de la page.
  - Le type d'encodage du contenu.
- **L'enveloppe SOAP (*SOAP Envelope*):** L'élément *Envelope* est obligatoire et sert de conteneur aux autres éléments du message SOAP, c'est la racine du document XML. Cette enveloppe est constituée de :

  - **L'entête SOAP (*SOAP Header*) :** L'élément Header est optionnel mais s'il est présent, il vient immédiatement après l'élément *Envelope*. Il permet d'intégrer au message SOAP des directives lui permettant d'externaliser certains services. Ainsi, le message contiendra différentes instructions destinées à être prises en charge par des services extérieurs.
  - **Corps SOAP (*SOAP Body*) :** L'élément Body Constitue un conteneur pour la partie utile destiné au destinataire ultime du message. Il contient les données spécifiques à l'application. Il fournit un mécanisme simple pour échanger l'information avec le destinataire final du message. Il est utilisé pour effectuer des appels de procédures à distance (RPC) ou transporter le report des erreurs.

I.1.7.2.3. Formats de message SOAP

Il existe deux formats de messages SOAP correspondants respectivement aux messages sans pièces jointes et ceux qui peuvent en disposer [44].

- **Messages SOAP sans pièces jointes :** Un message SOAP sans pièces jointes comporte trois parties :
  - l'enveloppe (obligatoire) – *Envelope* – contient le nom du message suivi d'un domaine de nom (*namespace*) qui indique la version du schéma XML-SOAP supporté.
  - L'entête (optionnel) – *Header* – est utile quand le message doit être traité par plusieurs intermédiaires (chaîne de message).
  - Le corps (obligatoire) – *Body* – renferme les méthodes et paramètres qui seront exécutés par le destinataire final.
- **Message SOAP avec pièces jointes :** Le message peut comporter dans ce cas une ou plusieurs parties d'attachements ajoutées au message SOAP de base, la partie SOAP peut ne pas contenir seulement que le contenu XML, comme résultats, si un contenu de message n'est pas au format XML, il doit apparaître dans la partie *attachement*. Ainsi, par exemple, si votre message contient un fichier image, vous devez avoir une partie *attachement* pour le véhiculer dans un message SOAP.

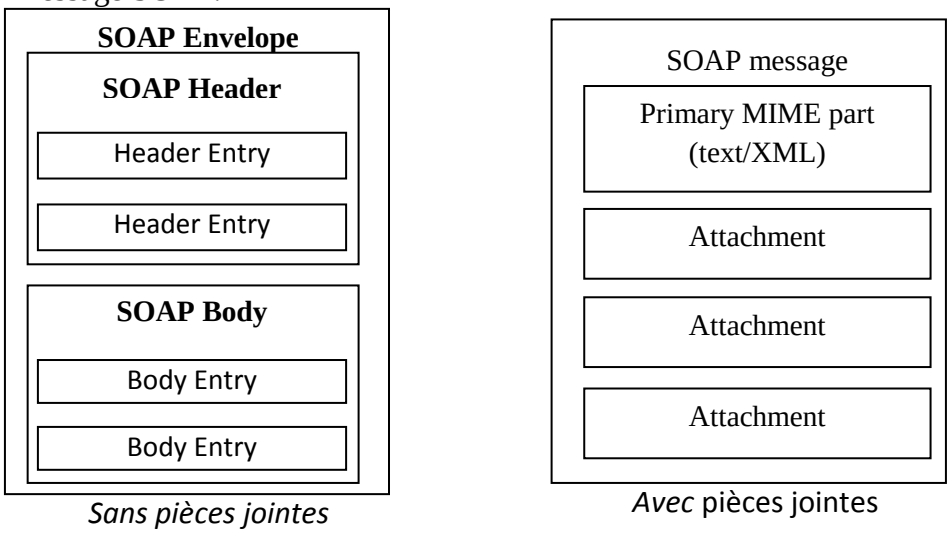


Figure I. 5. Format d'un message SOAP

La spécification SOAP 1.1 avec attachements a été soumise au W3C comme base du XMLP (*XML Protocol*). Elle utilise le MIME[1] comme *container* pour l'enveloppe SOAP et tous les attachements complémentaires.

I.1.7.3. La description des services web

Plusieurs langages de description sont apparus afin de décrire les services web, cela pour faciliter la publication, la recherche et la composition des service. Nous décrivons dans ce qui suit deux langages, WSDL créé par le W3C et DAML-S, conçu par le DARPA (*Defense Advanced Research Projects Agency*).

I.1.7.3.1. WSDL

WSDL (*Web Service Description Language*) est le standard actuel pour décrire les modalités de communications avec les services web et qui définit un cadre extensible pour décrire les interfaces des services web. C'est un langage basé sur XML. WSDL définit les activités du service web appelées *opérations,* et les données transmises appelées *messages*.

WSDL a été développé principalement par Microsoft et IBM et il est l'objet d'études d'un groupe de travail, le *Services Web Description Working Group*.
WSDL fournit une manière commune dont laquelle sont représentés les types de données passées dans les messages, les opérations à exécuter sur les messages. En outre, il définit les modalités techniques permettant de communiquer avec le service décrit. C'est un langage nécessaire au protocole SOAP.

Les éléments de WSDL

WSDL possède sept éléments distincts devisés en deux parties. Une première partie réutilisable, appelée *abstraite*, décrit le service. Une seconde partie non réutilisable, appelée *concrète,* indique la localisation du service [55].

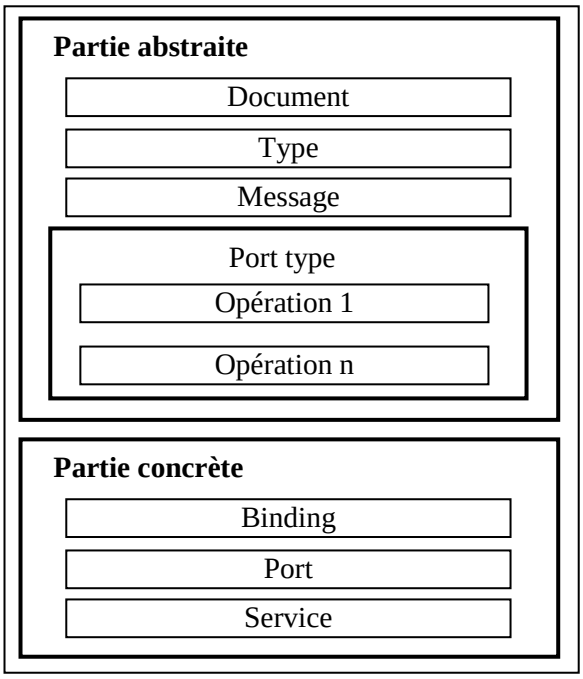


Figure I. 6. Structure d'un document WSDL

[1] Multi-purpose Internet Mail Extension

- **Partie abstraite :**

Elle comporte les éléments : document, type, message, portType

➢ **Document :** l'élément *wsdl:documentation* permet d'insérer des commentaires destinés à documenter un document WSDL. C'est un élément facultatif.

➢ **Type :** l'élément *type* permet de définir les structures de données contenues dans les messages échangés.

➢ **Message :** représente une définition abstraite de la donnée en cours de transmission qui sont les messages échangés.

➢ **portType :** le *portType* définit les traitements et les opérations sur un service web. Une opération représente une méthode. Chaque opération est identifiée par l'élément *opération* et ces opérations sont regroupées dans un élément *portType.*

- **Partie concrète :**

La section non réutilisable est composée des éléments suivants: binding, port et services, concrétisant les données abstraites de la première partie du document WSDL.

➢ **binding :** il décrit le mode d'encodage et définit les protocoles de communication utilisés lors des appels des services web.

➢ **Port :** spécifie une adresse pour une liaison définissant un simple point terminale de communication.

➢ **Service :** utilisé pour décrire les points d'accès (attribut port) du service web.

Les standards actuels des services web telles que WSDL, ne sont pas assez riches pour décrire sémantiquement ces services et ils limitent le mécanisme de découverte des services web seulement à la recherche basée sur mots clés, ainsi il est nécessaire d'enrichir la description des services web en donnant des informations sémantiques.

I.1.7.3.2. OWL-S

OWL-S (*Ontologie Web Language Service*) est un langage de méta-description de services. Il utilise la logique de description et les ontologies définies par OWL [15] (*Ontology Web Language*), langage de définition d'ontologies. OWL-S prend la suite de DAML-S (*DARPA Agent Markup Language Service*). Il définit un ensemble de classes et de propriétés afin de constituer l'ontologie. Les intérêts liés à l'utilisation de OWL-S sont que ce langage inclut la sémantique et contient des fonctions indispensables à la mise en œuvre des services web : la description, la recherche et l'invocation de service. OWL-S est composé de trois parties principales décrites ci-dessous :

I.1.7.3.2.1 Profil de service (*Service Profile*) :

Ce profil donne une description de haut niveau du service et de son fournisseur. Il est utilisé lors de la publication et la recherche d'un service. Il inclut trois types d'information décrites au format RDF (*Resource Description Framework*) : la description, le comportement fonctionnel et les attributs fonctionnels.

- *La description* doit être compréhensible par les humains. Elle est composée de la classe « Acteur » (*Actor*), dont les propriétés sont les suivantes : adresse physique (*physical address*), URL du site Internet (*WebURL*), nom (*name*), téléphone (phone), adresse électronique (*email*), et fax. Deux classes héritent de la classe « Acteur » : les classes du fournisseur de service (*Service-Provider*), et du demandeur de service (*Service-Requester*).

- *Le comportement fonctionnel* permet de rendre public les opérations qu'effectue le service, en décrivant les paramètres d'entrée et de sortie de la méthode.

- *Les attributs fonctionnels* apportent des informations supplémentaires concernant le service. Ces informations peuvent être soit des pré-conditions à l'accès du service (par exemple : le client doit appartenir à la zone X), soit des données relatives à la qualité du service (temps de réponse, coût du service, etc.).

I.1.7.3.2.2. Modèle de service (*Service Model*) :

Le modèle de service est également écrit en RDF. Il présente le fonctionnement du service et définit les différents processus de service web (il s'agit du modèle de processus - *ProcessModel*). Les structures de contrôle et le flux de données véhiculées sont décrits. Le modèle de processus est représenté par la classe «*ProcessModel* ». Cette dernière possède comme propriétés : les paramètres d'entrées et sorties (nombre, type, etc.), les participants au processus, les pré-conditions et les effets du service. Il existe trois types de processus : le processus *atomique* directement invoqué par l'intermédiaire d'un accès (utilisation du *grounding profile*), le processus *simple* non directement invoqué, fournit simplement une vue d'un processus atomique ou la représentation simplifiée d'un processus composé, le processus *composé* décomposable en d'autres processus. Les composants principaux du modèle de processus sont l'ontologie du processus (*Process Ontology*) et l'ontologie de contrôle du processus (*Process Control Ontology*).

- **L'ontologie du processus** décrit un service en termes de paramètres d'entrée et de sortie, de préconditions, d'effets du service et, dans le cas approprié, des composants du sous-processus.
- **L'ontologie de contrôle** du processus décrit chaque processus comme étant un état, en prenant en compte son activation, son exécution et sa terminaison.

I.1.7.3.2.3. Accès au service (*Service Grounding*) :

L'accès au service est décrit par l'intermédiaire du protocole à utiliser, les formats de messages et le type de transport. Le contenu abstrait des messages échangés est décrit en tant que propriétés d'un processus atomique (dans le modèle de service). La spécification concrète des messages est effectuée par l'intermédiaire du langage WSDL.

I.1.7.4. La recherche des services web

Une fois le service web est définit et décrit en terme de mise en œuvre, et après la mise en place d'un protocole de communication, le service peut être déclaré dans un annuaire, on parle alors de la publication du service. Le service web doit être référencé afin de pouvoir être retrouvé et utilisé par une autre organisation ou un autre service.
Pour cela il existe de nombreux annuaires pouvant être soit internes à l'organisation, soit universel tel qu'UDDI (*Universal Description, Discovery and Integration*) et ebXML *(Electronic Business using eXtensible Markup Language),* registre consacrant au e-commerce.

I.1.7.4.1. EbXML

EbXML (*Electronic Business using eXtensible Markup Language*) est issu des travaux de OASIS [1](*Advancing Open Standards for the Information Society*). Il s'agit d'une structure permettant à toute entreprise de gérer ses transactions avec ses partenaires commerciaux via Internet. Son but est de standardiser le e-commerce, particulièrement le B2B. Le registre se base sur le méta-langage XML.

[1] http://www.oasis-open .org/home/indexe.php *dernière consultation 20/05/2009*

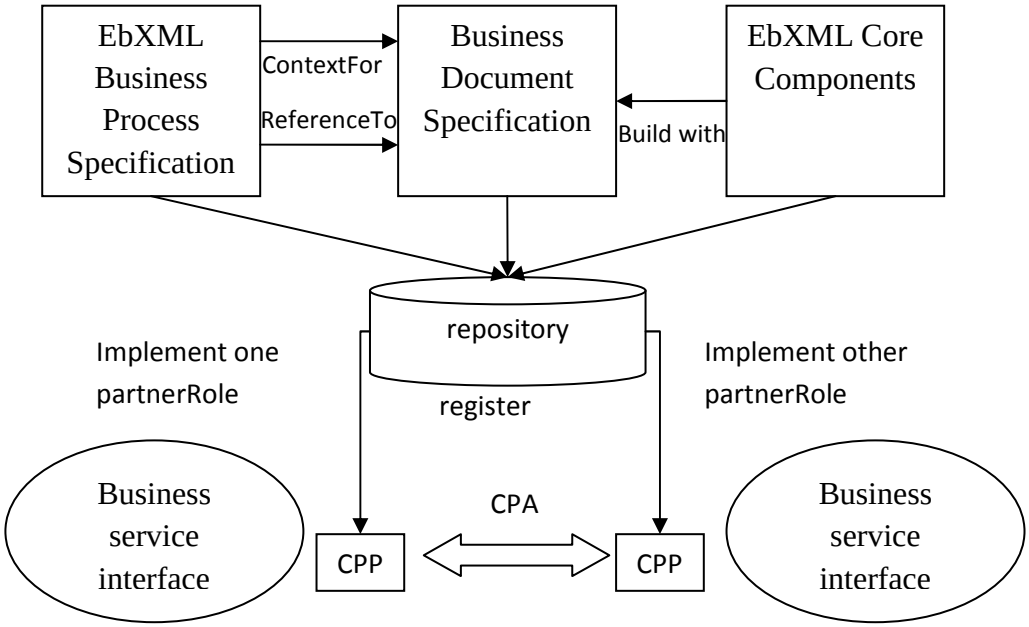


Figure I. 7. Architecture de EbXML

Chaque organisation voulant utiliser le registre ebXML doit posséder un profil de protocole de collaboration (CPP - *Collaboration Protocol Profile*). Ce profil contient les informations concernant les partenaires, leurs rôles, les échanges de documents en général, les contraintes de sécurité, etc.
Le lien entre deux partenaires est constitué par l'intermédiaire de l'entente de protocoles de collaboration (CPA - *Collaboration Protocol Agreement*). Il permet de formaliser l'accord et les interactions entre les organisations.
Les organisations ont pu avoir connaissance l'une de l'autre par l'intermédiaire du référentiel (Repository). Ce registre est un entrepôt de fichiers utilisés comme méta-données.

I.1.7.4.2. UDDI

UDDI (*Universal Description, Discovery and Integration*) est un standard, né de l'initiative d'un certain nombre d'entreprises, dont notamment Microsoft, IBM (*International Business Machines*), Sun (*Stanford University Network*), Compaq (*COMPatibility And Quality*), HP (*Hewlett Packard*), Intel (*Integrated Electronics*) et bien d'autres qui se sont réunies pour développer une spécification basée sur des technologies standard afin de faciliter la collaboration entre partenaires dans le cadre des échanges commerciaux.
UDDI est une spécification définissant la manière de publier et de découvrir les services web sur réseau, ainsi, lorsque l'on désire mettre à disposition un nouveau service, on crée un fichier appelé *business registry* qui décrit le service en utilisant le langage *XML schéma*, il contient des informations sur les entreprises et les services qu'il publie. L'inscription d'une société à l'annuaire UDDI lui permet de se présenter et présenter ses services.
L'enregistrement des services web dans un annuaire UDDI s'effectue auprès d'un opérateur en accédant au site web de ce dernier. L'annuaire UDDI facilite la localisation d'un service web et l'agrégation des services web émanant d'entreprises différentes. Par contre, il ne permet pas de localiser des partenaires (seulement des services) et n'exonère pas les entreprises d'instaurer entre elles une relation de confiance préalable à l'utilisation réciproque de leurs services web.
UDDI repose sur le protocole de SOAP et assure que les requêtes et les réponses sont des objets UDDI envoyés sous tortue de messages SOAP.

I.1.7.4.2.1. Les différents rôles d'UDDI

UDDI fournit trois services de bases [51]:

- **Publish:** ce service gère comment le fournisseur de service web s'enregistre lui-même ainsi que ses services (en utilisant UDDI).
- **Find:** ce service gère comment un client peut localiser le service web désiré (cela peut passer par des invocations de services web pour une utilisation automatique par un programme ou par une consultation d'annuaire en utilisant des mots clés).
- **Bind:** ce service gère également comment un client peut se connecter et utiliser le service web une fois celui-ci localisé.

I.1.7.4.2.2. Les structures de données UDDI

L'annuaire UDDI est consultable sous plusieurs facettes:

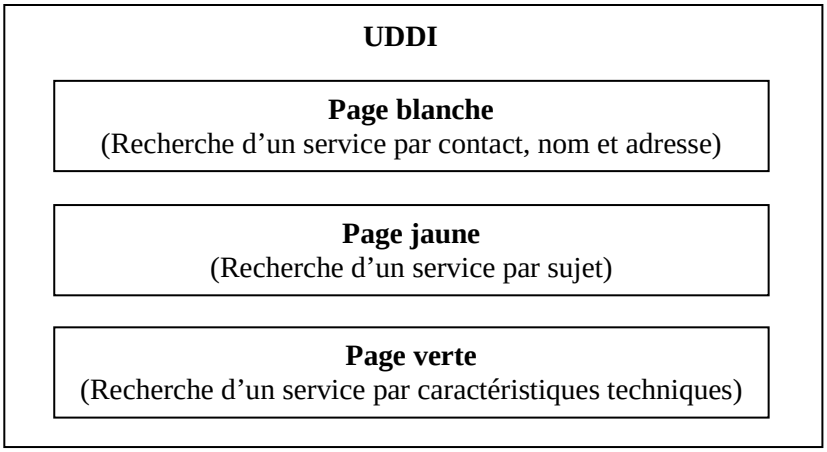


Figure I. 8. Annuaire UDDI

- ***Les pages blanches (white pages)*** recensent les entreprises ; elles comportent les informations sur les entreprises telles que le nom de l'entreprise, ses coordonnées et des descriptions de l'entreprise consultables par l'utilisateur.
- ***Les pages jaunes (yellow pages)*** comprennent la description, au format WSDL, des services web déployés par les entreprises. elles répertorient les services web par catégorie et regroupent les informations à propos de la classification des entreprises.
- ***Les pages vertes (green pages)*** fournissent des informations techniques détaillées sur les services fournis. Ces pages contiennent des informations sur les processus métier, des descriptions de services et des informations de liaison sur les services.

UDDI fournit aux clients des mécanismes de localisation dynamique d'autres services web. Un *UDDI Registry* a deux types de clients :

- ceux qui publient les services, les fournisseurs de services ;
- et ceux qui utilisent ces services.

I.1.8. Conclusion

Le web actuel est essentiellement *syntaxique*, dans le sens que la structure des documents (ou ressources au sens large) est bien définie, mais que son contenu reste quasi inaccessible aux traitements machines. Seuls les humains peuvent interpréter leurs contenus. La nouvelle génération de web (Le web sémantique) a pour ambition de lever cette difficulté. Les ressources du web seront plus aisément accessibles aussi bien par l'homme que par la machine, grâce à la représentation *sémantique* de leurs contenus.

I.2. Le web sémantique

Le web est constitué par un ensemble de documents, principalement textuels, formatés dans un langage particulier (HTML). Les logiciels possèdent de majeures difficultés en ce qui concerne la compréhension du contenu des pages web avec ce formalisme. Le web sémantique (WS) a été apporté pour résoudre ce problème.

I.2.1. Principes généraux du web sémantique

On peut définir le WS comme étant une infrastructure destinée aux machines plutôt qu'une application ou un langage bien déterminé. Les bénéfices attendus d'un web dont le contenu est accessible par les machines sont très divers à savoir:

1) La résolution des difficultés de recherche de l'information sur le web.
2) mise en relation des ressources (pages web…) d'une manière automatique.
3) raisonnement sur les annotations pour permettre une recherche d'information plus intelligente sur le web.

L'objectif primordial du WS est de rendre ces ordinateurs capables de comprendre les informations sans l'intervention de l'homme, afin d'accomplir les tâches fastidieuses assurées par ce dernier.

I.2.2. Les langages pour le web sémantique

I.2.2.1. Les langages du w3c (architecture en couche)

Le diagramme qui résume les principes sur lesquels repose le web sémantique est sous forme d'une architecture en couche connue sous le nom « *semantic Web layer cake* », comme montre la figure suivante [9].

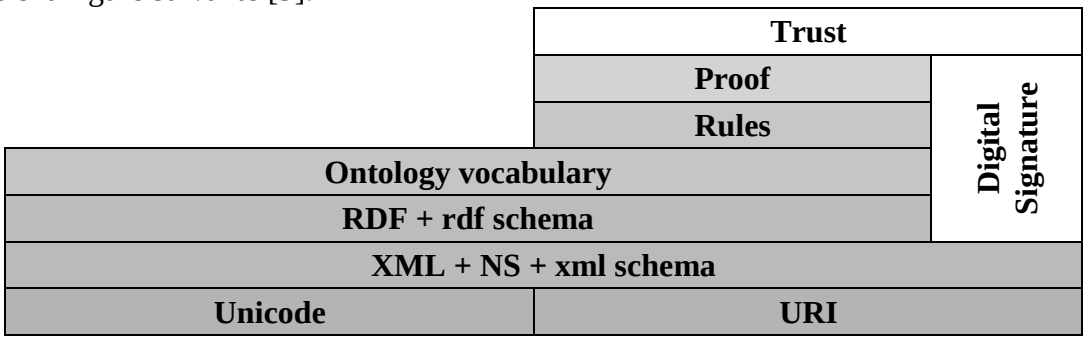


Figure I. 9. Architecture en couche du web sémantique

Du bas vers le haut, on y voit la graduation progressive du niveau symbolique au niveau connaissance :

– Le standard **Unicode** définit l'ensemble des caractères constituant l'alphabet

– **XML, XML Schema** et le mécanisme de **namespace** (NS) permettent de construire des documents structurés et de définir leur syntaxe.

– à partir de la couche **RDF + RDF Schema** apparaissent les langages de représentation des connaissances et la notion de sémantique est introduite.

– Au niveau de la couche **Ontology vocabulary**, les concepts de domaines de connaissances peuvent être représentés dans des ontologies formelles avec les langages OWL (*Ontology Web Language*).

– Les couches **Proof** (*preuve*) et **Trust** (*confiance*) constituent des objectifs non encore totalement atteints par la communauté du web sémantique.

I.2.2.2. RDF et RDFS

RDF (*Resource Description Framework*) est un modèle de données (ou un langage d'assertion et d'annotation destiné à exprimer les propositions en utilisant des vocabulaires

formels et précis), associé à une syntaxe, dont le but est de permettre à une communauté d'utilisateurs de partager les mêmes métadonnées pour des ressources partagées. Il a été conçu initialement par le W3C pour permettre de décrire l'information accessible sur le web et établir des relations entre les **ressources**. Il est donc particulièrement adapté aux annotations associées aux ressources du web.
RDF n'est pas particulièrement conçu pour permettre de stocker les métadonnées de documents mais plutôt pour permettre leur échange et leur traitement par des agents humains ou logiciels. Un des gros avantages de RDF est son extensibilité, à travers l'utilisation des schémas RDF qui peuvent s'intégrer et ne s'excluent pas mutuellement grâce à l'utilisation du concept d'espace de nom (*namespace*).
Un document RDF est composé d'un ensemble de triplets <*sujet*, *prédicat*, *objet*> [10], chaque élément du triplet pouvant être un **URI**, un littéral ou une variable.

I.2.2.3. Les langages de description

I.2.2.3.1. OWL

OWL (*Ontology Web Language*) est un langage de définition d'ontologies destiné, en particulier, à décrire les ressources du web.
OWL peut être utilisé pour représenter le sens des termes dans des vocabulaires ainsi que les relations qui existent entre ces termes. Cette représentation des termes et de leurs interrelations est appelée ontologie. OWL décrit la structure d'un domaine en termes de classes et de propriétés comme les approches orientées objets. Il facilite l'interopérabilité, au niveau machine, du contenu du web plus que ce qui est déjà supporté par XML, RDF et *RDF Schema* (RDF-S) en fournissant du vocabulaire supplémentaire avec des sémantiques formelles, qui indique comment déduire les conséquences logiques d'une ontologie, c'est-à-dire les faits qui ne sont pas littéralement présents dans l'ontologie mais peuvent être déduits par la sémantique formelle. Concrètement, la sémantique formelle d'un langage comme OWL peut être vue comme un ensemble de règles génériques de raisonnement (par exemple transitivité de la relation de *subsumption*).
OWL est le successeur de DAML+OIL, langage issu de la collision entre DAML et OIL et s'inspirant des principes de frames et de logique de description. OWL peut-être vu comme une extension de RDFS, mais auquel on aurait enlevé des propriétés telles que la possibilité de traiter les assertions comme des ressources [62].
OWL Permet à une ontologie d'augmenter le sens du vocabulaire prédéfini (RDF et OWL). De ce fait, OWL fournit des sous langages de plus en plus expressifs conçus pour faire un compromis entre son pouvoir expressif et son pouvoir de raisonnement.

– Le langage **OWL Lite** concerne les utilisateurs qui ont principalement besoin d'une hiérarchie de classifications et de fonctionnalités de contraintes simples. Par exemple, OWL Lite ne permet que des valeurs de cardinalité de 0 ou 1.

– Le langage **OWL DL** (*Description Logic*) concerne les utilisateurs qui souhaitent une expressivité maximum sans perdre la complétude du calcul (garantie de calculer toutes les conclusions) et la décidabilité (tous les calculs seront terminés dans un temps fini) des systèmes de raisonnement. Le langage OWL DL inclut toutes les structures de langage de OWL, mais sont utilisables avec des restrictions (par exemple, lorsqu'une classe peut être une sous classe de plusieurs autre classes, une classe ne peut être une instance d'une autre classe). OWL DL est ainsi nommé en raison de sa correspondance avec la logique descriptive, possède des propriétés de calcul avantageuses pour les moteurs d'inférences.

– Le langage **OWL Full** est destiné aux utilisateurs qui souhaitent une expressivité maximale et la liberté syntaxique de RDF sans garantie de calcul. Par exemple, dans OWL Full, une classe peut se traiter simultanément comme une collection d'individus et comme un

individu à part entière. OWL Full permet aussi à une ontologie d'augmenter le sens du vocabulaire prédéfini (RDF et OWL).

I.2.2.3.2. UDDI

Le protocole **UDDI** (*Universal Description, Discovery and Integration* [20] est une plate-forme destinée à stocker les descriptions des services web disponibles, à la manière d'un annuaire de style « Pages Jaunes ».

I.2.2.3.3. WSDL

WSDL [24] est un langage basé sur XML servant à décrire les interfaces des services web, c'est-à-dire en représentant de manière abstraite les opérations que les services peuvent réaliser, et cela indépendamment de l'implémentation qui en a été faite.

I.2.2.3.4. DAML-S

DAML-S [3] est un langage de description de services basé sur XML utilisant le modèle des logiques de descriptions (et plus précisément DAML+OIL).

I.2.2.3.5. XL

XL est une plate-forme destinée aux services web, basée sur XML, utilisant un langage propre de haut niveau *XL* (*eXpérimental Language*), XL permet une composition facilitée des services. Cependant, même si XL permet de manipuler facilement des services web, il ne permet pas de les décrire autrement que par des entrées/sorties XML, et la sémantique est absente, contrairement à DAML-S par exemple.

I.2.2.3.6. XDD

XDD (*XML Declarative Description*) est un langage capable de décrire toute la sémantique d'une ressource web en ajoutant un langage déclaratif à la syntaxe de XML.
Une description utilisant XDD est un ensemble d'éléments XML classiques, d'éléments XML étendus à l'aide de variables, et de relations entre les éléments XML sous forme de clauses. Un élément XML classique représente une unité sémantique et peut se substituer à un objet (au sens large) du domaine d'application. Un élément étendu, lui, permettra de représenter une information implicite ou un ensemble d'unités sémantiques. Les clauses peuvent exprimer des règles, des relations conditionnelles, des contraintes d'intégrité, et des axiomes ontologiques.

I.2.3. Les ontologies

I.2.3.1. Définition

Les ontologies sont un des concepts de base du web sémantique, leur but est l'étude des catégories des objets qui existent ou qui doivent exister dans un certain domaine [57]. La première définition de l'ontologie dans le domaine de l'informatique est proposée par *Neches et al.*, Ils définissent l'ontologie comme : *"les termes et les relations de base comportant le vocabulaire d'un domaine aussi bien que les règles pour combiner les termes et les relations afin de définir des extensions du vocabulaire"*. Les ontologies sont étudiées dans le domaine de l'intelligence artificielle. Elles permettent la représentation des connaissances, et elles sont aussi évoquées dans le web sémantique. Une définition consensuelle, précise et complète des ontologies dans le contexte du web sémantique n'existe pas encore [5]. Cependant, *Gruber* en 1993 proposait la définition la plus citée. Il définit l'ontologie comme étant *"une spécification explicite d'une conceptualisation"* [26]. La conceptualisation est le résultat d'une analyse du domaine étudié et l'abstraction du monde de ce domaine. Cette conceptualisation est représentée dans une forme concrète où les concepts, les relations ainsi que les contraintes sont explicitement définies dans un format et langage formel.

Partant du fait que plusieurs connaissances peuvent prendre des représentations différentes, on trouve plusieurs ontologies de domaine pour un même champ d'application. Il est alors nécessaire de disposer d'outils permettant de faire le lien entre les connaissances exprimées dans chacune des ontologies. Le web sémantique exploite les ontologies dans la représentation des connaissances. Cette exploitation est essentielle dans la réutilisation des ontologies dans des systèmes permettant leur manipulation.

I.2.3.2. Langages de représentation d'ontologies

Un langage ontologique est un langage formel permettant de représenter les différents éléments constituant une ontologie.
Plusieurs langages sont proposés pour la description des ontologies tel que : RDF(S) (RDF et RDF Scheme), DAML+OIL [19] et OWL (*Ontology Web Language*). Une ontologie permet de décrire des connaissances d'un domaine. Il semble intéressent de représenter l'ontologie dans un langage expressif. Quelques langages sont expliqués ci-dessous :

- **KIF** (*Knowledge Interchange Format*): est un langage désigné pour les échanges des informations entre les systèmes. Il utilise une syntaxe tel que *Lisp*, c'est un langage de bas niveau pour la représentation des ontologies.
- Le langage **RDF(S)** [37] ne permet pas représenter la cardinalité d'une relation. Il n'exprime pas aussi les caractéristiques des relations : la transitivité, la symétrie, la fonctionnalité, etc.
- **DAML (***DARPA Agent Markup Language***)** qui est une extension de XML et RDF.
- **DAML + OIL** est une amélioration de DAML qui fournit un ensemble riche de constructeurs pour créer les ontologies.
- Le langage **OWL** qui est le langage d'ontologie spécialement développé pour la représentation des ontologies dans le cadre du web sémantique. Ce langage permet aussi de créer, partager et échanger des connaissances dans le web sémantique.
- **XOL (** *XML-based Onrology Exchange***)** [33] : il est utilisé pour la création des ontologies partagées et destiné pour être un langage intermédiaire pour le transfert des ontologies entre les bases de données du système et les programmes d'application.

I.2.3.3. Les principes de construction des ontologies

Certain nombre de travaux proposent des principes de construction d'ontologies :

- **Clarté.** Les ambiguïtés doivent être réduites.
- **Cohérence.** Une ontologie doit être cohérente.
- **Extensibilité.** L'ontologie doit être construite de telle manière que l'on puisse l'étendre facilement, sans remettre en cause ce qui a déjà été fait.
- **Biais d'encodage minimal.** L'ontologie doit être conceptualisée indépendamment de tout langage d'implémentation.
- **Engagement ontologique minimal.** Une ontologie doit faire un minimum d'hypothèses sur le monde : elle doit contenir un vocabulaire partagé mais ne doit pas être une base de connaissances comportant des connaissances supplémentaires sur le monde à modéliser.

La conception d'une ontologie, comme la conception d'une base de données s'appuie sur des méthodes, des modèles, des outils et des langages. Les langages permettant de représenter des ontologies sont distingués en langages traditionnels (les frames, post-frames, logiques de description, orientés objet,…) et les langages orientés ressources tels que **XML** (*eXtensible Markup Language*) [12], **XOL** (*XML-based Onrology Exchange*) [47], **RDF**, **RDFS** (*Ressources Description Framework - Sheme*), **OIL** [51] ou **DAML+OIL** [30].

I.2.4. conclusion

L'ingénierie des ontologies est devenue une technologie importante pour la découverte sémantique, l'interopérabilité et l'intégration des sources hétérogènes. Cette technologie devient la base du web sémantique. DAML-S est un bon exemple montrant comment les ontologies révolutionnent le web. Le domaine du web sémantique est en développement continu. Des travaux de standardisation sont en cours afin de répandre l'approche sémantique sur le web entier. L'objectif est de rendre le web de demain plus intelligent que le web actuel, pour cela beaucoup de recherche et de développement sont à réaliser dans le cadre de ce domaine.

I.3. Les services web sémantiques

La notion du domaine des services web désigne essentiellement une application mise à disposition sur Internet par un fournisseur de service, et accessible par les clients à travers des protocoles Internet standards.
Le domaine du web sémantique a pour objectif de définir une extension du web actuel dans lequel non seulement l'information mais aussi la sémantique de cette information est décrite dans les documents.
A la convergence de ces deux domaines de recherche dans les technologies de l'internet, on trouve *les services web sémantiques*, qui a comme objectif de créer un web sémantique de services possédant une description non ambigüe de ses interfaces et ses propriétés et pouvant exploiter ces dernières en utilisant des couches conceptuellement indépendantes.
Deux approches pour ajouter de la sémantique à la description des services web sont apparues : La première est d'employer un nouveau langage de description sémantique comme OWL-S, mais la question qui s'est posé est pourquoi ne pas utiliser les standards existants (WSDL et UDDI) en leur ajoutant de la sémantique au lieu de créer un nouveau langage plus complexe et non standard, ce qui a donné naissance aux annotations sémantiques comme deuxième approche de description sémantique dans le domaine des services web.

I.3.1. Annotations Sémantiques

Une annotation sémantique est l'information additionnelle dans un document qui définit la sémantique d'une partie d'un document. Dans le contexte des services web, les annotations sémantiques sont les éléments de l'information additionnelle dans un fichier WSDL. Ils définissent la sémantique en se rapportant à une partie d'un modèle sémantique qui décrit la sémantique de la partie du fichier étant annoté.
L'ajout de la sémantique aux standards des services web ou l'annotation sémantique des services web se résume en trois étapes [56]:

- Ajout de la sémantique à WSDL en utilisant des ontologies.
- Utiliser l'UDDI pour stocker les annotations sémantiques des services web basées sur les ontologies partagées et pour la recherche des services web basée sur ces annotations.
- Proposition d'un algorithme pour la découverte sémantique des services web, qui emploie la fonctionnalité du service web comme critère principale pour la recherche.

D'un point de vue technique, la description d'un service web inclut tous les détails nécessaires à l'interaction avec le service comme, par exemples, le format des messages, les signatures des opérations, le protocole de transport et la localisation du service. Les services web s'appuient sur des mécanismes et des protocoles standards et sont indépendants des langages de programmation (Java, C++, Perl, C#, etc.), du modèle objet (COM, EJB, etc.) ainsi que des plates formes d'implémentation (J2EE, .NET, etc.).

I.3.2. L'architecture de service de Web d'IBM

L'architecture de service web d'IBM (*International Business Machines*) est représentée dans la figure suivante.

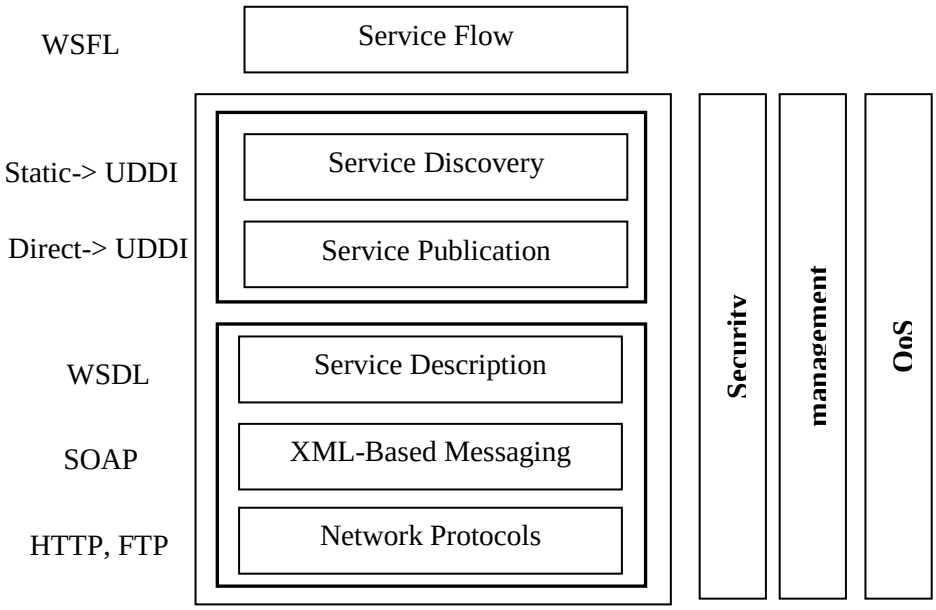


Figure I. 10.L'architecture des services web de IBM

- SOAP et les protocoles du réseau : Les couches inférieures sont basées sur les protocoles généraux du réseau, comme le HTTP ou le FTP (*File Transfer Protocol*), et le SOAP (*Accès Simple Protocol D'objet*).
- WSDL : L'interface d'un service web et ses interactions sont décrites par le langage de description de services web WSDL.
- UDDI : La *publication de service* est considérée comme une description liée au marché d'un service ; UDDI contient tous les mécanismes qui permettent au demandeur de service d'accéder à la publication de service et d'entretenir la description.
- WSFL : Prescrit un format XML pour spécifier la composition de services, WSFL **(***Web Services Flow Language*) permet de composer des services complexes à un service plus simple.

I.3.3. Modélisation des services web sémantiques

Les architectures conceptuelles des services web sémantiques schématisent la conception du service comme spécifications d'un ensemble de couches qui couvrent potentiellement toutes les dispositifs du service.
Ces dispositifs permettent à des programmes ou à des agents externes de découvrir, invoquer et composer de nouveaux services, sont les suivants :

- *Les dispositifs d'accès (ou communication)* décrivent le protocole de communication.
- *Les dispositifs descriptifs* détaillent les propriétés du commerce électronique d'un service web sémantique.
- *Les dispositifs fonctionnels* spécifient les possibilités (capacités) de service web sémantique, décrites en termes de leurs données d'entrée-sortie, des effets et pré/post-conditions de l'exécution.
- *Les dispositifs structuraux* décrivent la structure interne d'un service composé.

Typiquement, les agents emploieront ces dispositifs pour la composition de service, lorsqu'ils déterminent qu'il y a des interactions entre les services secondaires.

I.3.3.1. La description des services web sémantiques à base des ontologies

Cette approche a été suivie par quelques auteurs [2], ils emploient un langage de balisage riche de coté sémantique pour créer une ontologie (ex : OWL-S) qui décrit les dispositifs de service.
Le *SWS* doit être traduit à un langage orienté *SWS* tel que OWL-S. La figure suivante montre la pile d'ontologies qui décrit tous les dispositifs d'un *SWS.*

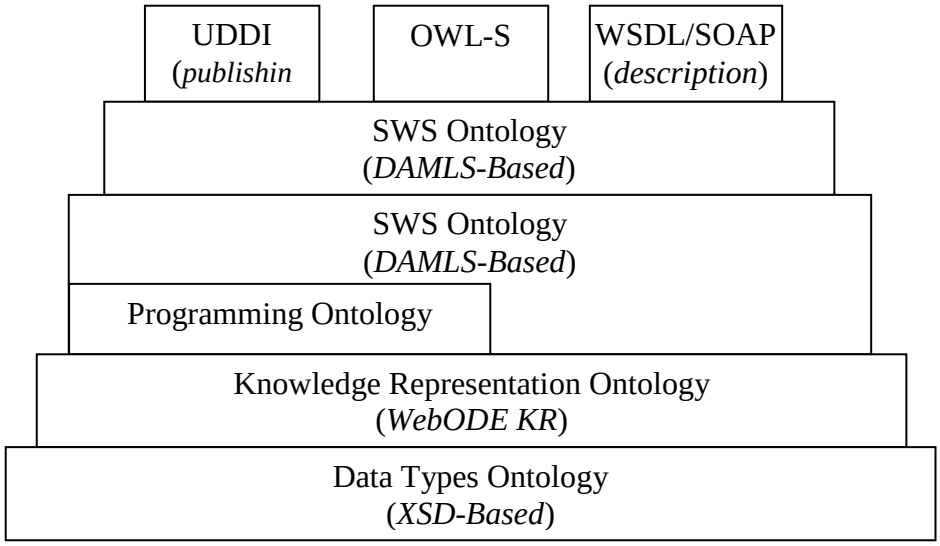


Figure I. 11. L'ensemble des ontologies définies dans la structure des services web sémantiques

La pile se compose d'ontologies suivantes :

1) une ontologie pour décrire les méthodes de résolution des problèmes qui seront employées pour représenter la structure interne et les dispositifs fonctionnels d'un SWS ;

2) une ontologie décrivant les concepts de niveau supérieur qui définissent les dispositifs d'un service de web sémantique ;

3) une ontologie pour définir les entités de représentation de la connaissance employées pour modeler un *SWS* et un domaine ontologie à un niveau de connaissance ;

4) une ontologie pour décrire les types de données à employer dans un domaine ontologie.

I.3.3.2. DAML-S

Parmi les recherches qui se fixent pour but d'enrichir l'approche des services web, on peut citer DAML-S qui est une ontologie DAML+OIL décrivant des services web et dont le but est d'automatiser la découverte, l'invocation, la composition et l'exécution des services web.
Comme l'ontologie de DAML+OIL, DAML-S bénéficie des avantages des contenus web décrits en DAML+OIL. Il a une sémantique bien définie et permet la définition d'un vocabulaire pour les contenus des services web en termes d'objets, de relations complexes, de classes et sous-classes, de restrictions sur les cardinalités, *etc.* Il inclut également toutes les informations entrées en XML.
DAML-S propose donc une ontologie de haut niveau des services web sous forme d'un ensemble de classes. La classe SERVICE, qui est le haut de l'ontologie DAML-S, se voit ainsi associer un ensemble de connaissances par l'intermédiaire de deux classes :

• La première SERVICEPROFILE fournit, par un ensemble d'attributs et de propriétés, l'information nécessaire pour qu'un agent puisse découvrir un service (ces attributs et propriétés sont par exemple les entrées du service, ses sorties, ce qu'il offre, ses préconditions et ses effets, *etc.*).

• La deuxième SERVICEMODEL décrit comment utiliser le service en des termes plus abstraits que WSDL (par un modèle de procédé du service : flot de données et flot de contrôle). Cette classe est désignée pour permettre la composition et l'exécution automatique des services

D'autres classes encore (comme SERVICEGROUNDING) sont associées à celles-ci, afin de donner des descriptions conceptuelles des services web permettant par exemple d'apparier une requête et une description de service en raisonnant à l'aide de l'ontologie.

I.3.3.3. WSFL

WSFL (*Service Web Flow Language*) est un langage pour la description de la composition des services web basé sur XML.

WSFL fournit un support pour la composition récursive des services. Dans WSFL, chaque composition de services web peut devenir elle-même un nouveau service web, qui peut être utilisé comme un composant pour une nouvelle composition.

I.3.4. Conclusion

Aujourd'hui, les services web sémantiques constituent une voie prometteuse permettant de mieux exploiter les services web en automatisant les différentes tâches liées au cycle de vie d'un service. Ils posent aujourd'hui un certain nombre de problèmes, qui interpellent différentes communautés de recherche, aussi bien théoriques qu'appliqués. Les travaux effectués dénotent une vitalité réelle de ce domaine de recherche émergent.

Cependant, la tendance actuelle des communautés de recherche s'intéressant aux services web sémantiques ne tient pas en compte les caractéristiques fondamentales des services web et de l'environnement dans lequel ils doivent s'intégrer, le succès de cette voie de recherche dépend de sa capacité et doit tenir compte des facteurs suivants :

- Les travaux de recherche devront intégrer le plus possible les caractéristiques des futurs standards actuellement en cours d'élaboration. Ils doivent donc s'efforcer d'exploiter et compléter ces futurs standards et non pas ignorer leur existence ou les concurrencer.
- La volonté d'automatiser à excès n'est certainement pas une voie réaliste. Certains travaux de recherche semblent faire abstraction de la complexité du contexte de l'intégration de par les hypothèses simplificatrices fortes qu'ils imposent dans leurs solutions.
- Le concept de sémantique tel que défini dans le contexte du web sémantique, i.e., décrire la sémantique de manière à la rendre intelligible pour les machines, semble trop limité.

CHAPITRE II

La découverte et la composition des services web

Chapitre II

La découverte et la composition des services web

II.1. Introduction

Les services web sémantiques sont un effort de recherches pour automatiser l'usage des services web, le composant nécessaire pour le web sémantique. Pour certaines raisons, la description sémantique détaillée et complète statique de service n'est pas faisable, le client (logiciel) ne peut pas choisir la meilleure offre de service pour un but d'utilisateur donné seulement en employant les descriptions statiques du service. Par conséquent, le client interagit automatiquement avec les services web découverts pour découvrir les offres, en d'autres termes pour trouver l'information nécessaire de choisir la meilleure offre qui accomplira le but de l'utilisateur.
Le web sémantique est non seulement une extension du web courant avec plus de descriptions sémantiques des données; il doit également intégrer les services qui peuvent être employés automatiquement par l'ordinateur à la place de l'utilisateur.
Afin de faire des services web une partie du web sémantique, le secteur de recherches des services web sémantiques (SWS) vise à augmenter le niveau de l'automation de certaines tâches, par exemple découvrant les services disponibles et les composants pour fournir des fonctionnalités plus complexes. L'automation de SWS est soutenue par des descriptions sémantiques du service web. La découverte sémantique de service web est le centre de notre travail.

II.2. La découverte des services web

II.2.1. Principes Généraux

La recherche de service confronte deux acteurs : le fournisseur ou producteur de service ; qui cherche à annoncer du mieux possible ses services et l'utilisateur ; qui ne sait pas où chercher le service.
Il faut donc mettre en place un agent faisant le lien entre ces deux acteurs et qui fournira les capacités suivantes :

1. Le fournisseur enregistre son service auprès de l'agent.
2. L'agent stocke cette information dans son registre, sa base de connaissance.
3. Le client envoie sa demande à l'agent.
4. L'agent répond au client par la liste des services satisfaisant sa requête.

L'information stockée par l'agent doit représenter les spécificités du service annoncé. Le langage de description de service donne les informations que le producteur doit fournir à l'agent.
Cette information doit être expressive autrement dit représenter toutes les informations intéressantes dans le processus de recherche de service et doit être facile à utiliser : lisible mais aussi facile à écrire par le fournisseur de service.
La demande du client peut prendre deux formes, selon le choix du concepteur de l'agent : Soit l'agent met à la disposition du client un mécanisme d'interrogation. La demande du client correspond à une interrogation du registre de l'agent. Soit le client exprime sa requête dans le langage de description de service et envoie cette requête à un agent qui fait correspondre la demande à l'offre via un algorithme de *matching*.
L'algorithme de *matching* compare toutes les annonces à la requête, pour fournir comme résultat les annonces qui sont proches de la requête.
L'algorithme permet généralement au client de fixer le degré de correspondance désiré entre sa requête et les annonces.
L'algorithme de matching doit permettre une recherche [31]:

- **précise**. Les résultats renvoyés par l'algorithme doivent correspondre aux définitions des degrés de correspondance.
- **efficace**. Le temps de réponse doit être le plus court possible.
- **efficiente**. L'ensemble des résultats ne doit pas être trop grand. Mieux vaut un petit ensemble avec un degré de correspondance élevé.

L'algorithme doit réduire le nombre de faux positifs et de faux négatifs.
De plus, il serait intéressant d'encourager les acteurs à faire une description honnête de leur service. Autrement dit, si la description n'est pas honnête, le service n'est pas ou mal repris dans les résultats lui correspondant.

II.2.2. Autour de la découverte des services web

Dans le contexte d'une application qui a besoin d'exécuter une fonctionnalité implémentée comme un service web par plusieurs fournisseurs, la découverte fait référence au processus de recherche des services web implémentant la fonctionnalité souhaitée. Les registres UDDI sont des entités qui servent d'appui à la découverte de services web pour les applications client. De cette façon une application interroge un registre UDDI pour les fournisseurs d'un service web. La meilleure manière de définir la découverte de services est en décrivant les problèmes qu'elle vise à résoudre. D'abord, passons en revue les fonctions distinguables d'un environnement sémantique d'exécution. Les étapes suivantes sont traditionnellement exécutées après qu'un utilisateur soumette son but.

1- **La découverte** en utilisant des descriptions publiées, trouve tous les services web disponibles (les services peuvent être plus génériques, ou plus spécifique).

2- **En filtrant :** filtrer les services qui n'adaptent pas les contraintes de l'utilisateur

3- **Ranger, sélection** de choix les offres restantes basées sur les préférences de l'utilisateur. Le service meilleur rangé peut être automatiquement choisi, ou le rang peut être présenté à l'utilisateur.

4- **l'invocation** utilise le service choisi de réaliser le but.

Dans ce sujet ils se sont trouvés trois travaux pertinents : OWL-S (Ontology Web Language Service), *Services Web Dynamic Discovery et Service Web Discovery Architecture*. Ces travaux sont décrits dans ce qui suit.

II.2.2.1. OWL-S

OWL-S (*Ontology Web Language - Service*) est un langage qui définit une ontologie de services web [13]. Il est basé sur le langage OWL. Il permet de réaliser les deux tâches suivantes :

1) Découverte automatique de services web : Cette tâche est possible parce que OWL-S permet d'exprimer et de résoudre des requêtes avec contenu sémantique.

2) Invocation automatique de services web : OWL-S fourni un ensemble de APIs pour que l'invocation d'un service web soit automatique.

OWL-S a trois parties principales :

• Le profil du service : Pour faire de la publicité et découvrir des services.

• Le modèle du processus : qui donne une description détaillé d'une opération du service.

• Le « *grounding* » qui fourni les caractéristiques techniques pour établir la communication avec le service au moyen de messages.

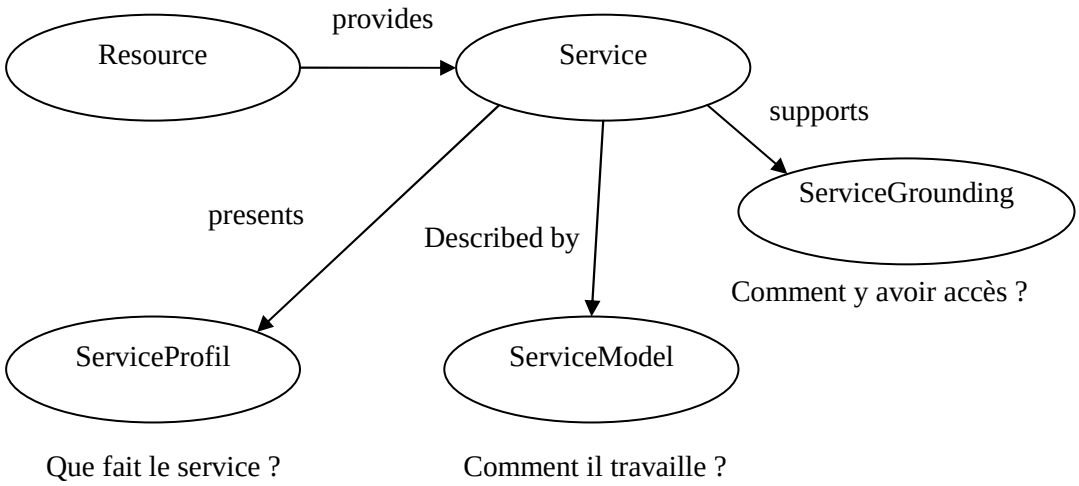


Figure II. 1. Niveau supérieur de l'ontologie OWL-S

II.2.2.2. Découverte dynamique des services web

Web Services Dynamic Discovery est une spécification développé par les représentants de *Microsoft, Bea Systems, Intel* et *Canon,* elle désigne la possibilité de localiser automatiquement un service web qui répond à des besoins particuliers. L'architecture de cette spécification propose une entité appelée Mandataire de Découverte (Figure suivante).

L'objectif de cette entité est de maintenir l'information des fournisseurs d'un ensemble de services web. Cette entité est disponible sur internet. Elle est localisée de manière indépendante, généralement éloigné, des applications client comme des fournisseurs de services web.

Lorsqu'un service web commence son exécution, il s'enregistre auprès le mandataire de découverte. Avant qu'un service finis son exécution, il manifeste ce fait au mandataire de découverte. En plus, quand une application client a besoin d'interroger les fournisseurs d'un service, elle envoie un message *multicast* à un ensemble de fournisseurs. Les fournisseurs qui donne le service répondent d'une façon affirmative à l'application et les autres, répondent d'une façon négative.

Si dans le processus décrit avant, l'application observe un mandataire de découverte, elle interrompt la communication avec les fournisseurs et commence une communication directe

avec le mandataire de découverte. L'application client le demande au mandataire de découverte sur les fournisseurs d'un service web spécifique.

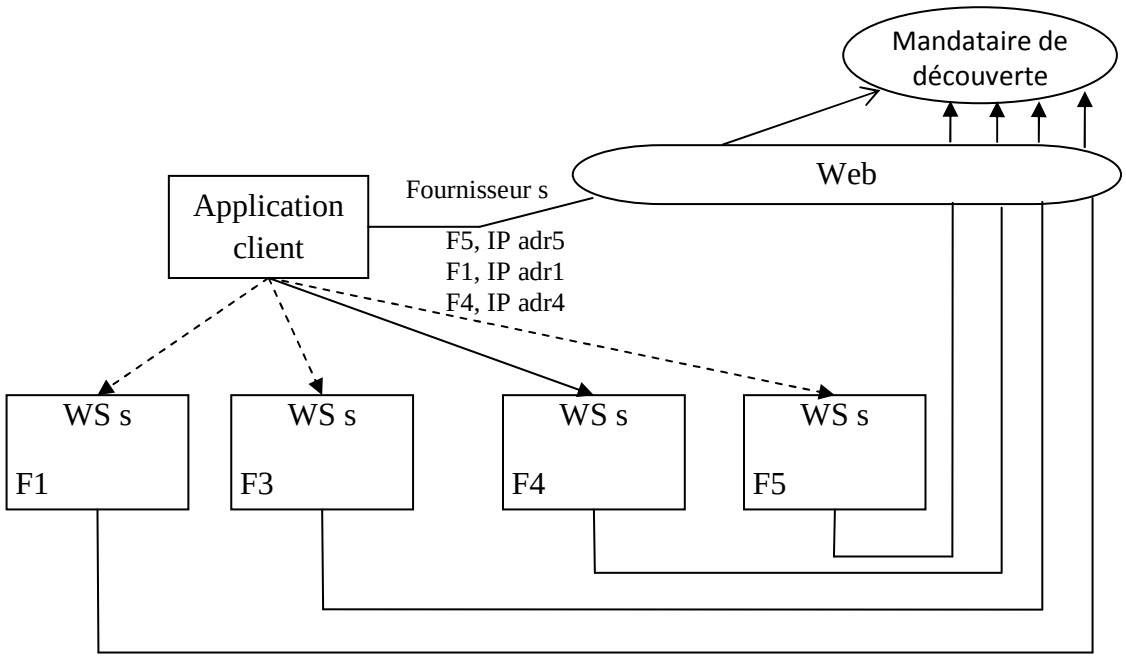


Figure II. 2. Architecture de Services Web Dynamic Discovery

II.2.2.3. L'architecture de découverte des services web

Sur ce problème il y a une approche qui s'appelle WSDA. WSDA (*Web Service Discovery Architecture*) est une architecture ouverte pour la découverte des services web. WSDA incorpore plusieurs standards comme XML, XML Schéma, SOAP, WSDL et XQuery. Celle-ci définit services, interfaces (ex. *consumer*), opérations (ex. *publish*) et sa liaison avec des protocoles.
Elle fournit quatre interfaces qui composent l'information nécessaire pour le registre d'un service web : *Presenter, Consumer, MinQuery et XQuery.* Une description plus détaille est la suivante:

- *Presenter :* Fournit au client la possibilité de chercher la description la plus récente du service web. Cette interface a un identificateur qui peut être un URI (*Uniform Resource Identifier*) ou un URL(*Uniform Resource Locator*) et un mécanisme de recherche.
- *Consumer :* Permet au fournisseur de publier le contenu du service pour être interrogé par le consommateur. Cette fonctionnalité est fournie en utilisant l'opération *publish*. Le contenu du service peut être la description du service ou de la qualité du service, un fichier, la localisation d'une copie etc.
- *MinQuery :* Fournit un support basique d'interrogation
- *XQuery :* Fournit un support pour l'interrogation des services en utilisant XQuery comme langage de requêtes. *query* est l'opération fournie par cette interface.

Dans WSDA une caractéristique très importante est de pouvoir faire des requêtes en utilisant le langage XQuery, puisque cela est beaucoup plus puissant que le type des consultations permises par l'API d'UDDI. La troisième caractéristique considère comme complète, très utile et facile à utiliser, la fonctionnalité offerte à travers des quatre interfaces.

II.2.3. La substitution des services web

Il y a besoin de la substitution d'un service web par un autre dans les cas suivants :

• Pendant l'exécution, le service web échoue ou ne peut pas être atteint.

• Il y a un nouveau service web qui fourni un service mieux, en ce qui concerne les paramètres de qualité de service.

• Une nouvelle version d'un service web choisi, offert par le même fournisseur est disponible.

Avec la spécification d'un processus coopératif, le programmeur d'application compose les services web disponibles, pour satisfaire les exigences données pour un processus coopératif. Considérant les services web comme des boîtes noires, deux comportements différents doivent être considérés pour le processus coopératif :

- Comportement non-observable : il est lié au comportement d'une composition des services web. Puisque on considère un service web comme une boîte noire, nous ne pouvons pas modifier sa structure interne, mais seulement utiliser les méthodes fournies, en utilisant ses spécifications.
- Comportement observable : il est lié à la coopération existante parmi les organisations qui participent au processus.

Avec la substitution, le but est d'être capable de substituer un service web, si c'est nécessaire, sans affecter le comportement observable du processus coopératif. L'approche présentée dans [4], au problème de la substitution, est basée sur l'idée que les spécifications de services web doivent être classées pour suivre à la trace les relations entre eux en termes de compatibilité.
Dans ce point de vue, les services web sont classés dans deux types :

- Des services web spécifiques pour un secteur : Orientés aux processus coopératifs du secteur.
- Des services web avec propos général : définis pour être utilisés dans beaucoup d'environnements.

Cette distinction est faite parce que le registre des services maniés dans le secteur est contrôlé. A chaque service web du secteur on lui exige son information syntaxique et son information sémantique.
La spécification des services web spécifiques de chaque secteur sont inscrits dans une version améliorée du registre UDDI, appelé registre VISPO. La connaissance sémantique de services et le contenu de l'information est organisée dans l'*Ontologie de Service du Domaine* et l'*Ontologie de Connaissance du Domaine*. Dans cette approche une partie de la spécification d'une méthode d'un service est la pré-condition et la post-condition. Ces deux assertions s'écrivent en utilisant la logique de premier ordre.
Quand les problèmes liés à la substitution surgissent (par exemple un service échoue), le service concret échoué peut être substitué avec un autre appartenant à la même classe selon la configuration d'informations.

II.2.4. Les approches existantes de découverte de service

II.2.4.1. la découverte traditionnelle de service

CORBA [46] a proposé une des premières approches de découverte de service. Il indique des services *naming* et *trading* [47] employés pour découvrir des objets sur un réseau. Le service *naming* est *keyword-based* tandis que le service *trading* supporte la découverte de service. *UDDI* est l'approche de découverte de service la plus utilisée pour des services web. Le noyau de l'architecture d'UDDI est un enregistrement central qui fonctionne comme service *naming* et d'annuaire. Des services dans cet enregistrement sont décrits par trois perspectives

différentes, pages blanches, jaunes et vertes. En outre, les descriptions de service se composent de tModels qui classifient le *business service* ou services web en utilisant les taxonomies standard ou définies par l'utilisateur. *OSGi* [48] propose une plateforme ouverte de service pour la distribution des applications et des services pour tous les types de dispositifs gérés en réseau. La découverte de service est effectuée en interogeant le nom ou le type d'un service. *OSGa* se concentre sur l'intégration des paradigmes de calcul de grille avec des technologies de services web. Le service est annonce par son information (c.-à-d. nom, type) dans l'enregistrement. En recherchant cette information, l'utilisateur peut découvrir des services.
Klein [36] discute plusieurs catégories des technologies de découverte de service et leurs limitations pour la qualité du résultat de la découverte de service. Selon les catégories de Klein, les approches traditionnelles de découverte de service *keywords-based* ou *tablebased* ne tiennent pas compte de l'information contextuelle. Ça mène à la mauvaise qualité des résultats recherchés.

II.2.4.2. découverte du contexte du service

Dans cette partie, nous présentons les approches existantes qui considèrent l'information contextuelle dans le processus de découverte de service et nous discutons également les problèmes d'employer l'information contextuelle dans ces approches.
Le projet de Cooltown [32] permet à des utilisateurs de découvrir les services qui sont à proximité de l'utilisateur. Dans cette approche, l'endroit de l'utilisateur et du service est employé pour dériver que l'utilisateur est dans l'endroit du service. De cette façon, les services qui sont près de l'utilisateur sont retournés par le mécanisme de découverte de service. *La plateforme pour les applications adaptatives* propose l'architecture pour les applications qui adaptent leur comportement selon le contexte de l'utilisateur. La plateforme permet la découverte des fournisseurs de contexte par le type de contexte qu'elles annoncent. Cette information contextuelle est employée pour adapter le comportement d'application. *Le projet* de CB-Sec [38] fournit la fonctionnalité pour découvrir les services qui sont à proximité de l'utilisateur. Cette approche tient compte des possibilités de l'utilisateur et du service dans le procédé de découverte de service.
L'information contextuelle est fortement interliée et elle a beaucoup de représentations alternatives. Ceci la rend difficile à l'interpréter et l'employer. Les fournisseurs de contexte et les consommateurs de contexte (par exemple les fournisseurs de service ou le demandeur) peuvent avoir différents compréhension de la même information contextuelle. Ceci mène à l'interprétation fausse de l'information, qui mène alternativement à la mauvaise compréhension du but d'utilisateur et donc des résultats faibles de découverte.

II.2.4.3. découverte de service basée sur les ontologies

Comme nous avons dit plus tôt, la compréhension partagée sur les concepts, employés pour décrire des services et l'information contextuelle, est cruciale d'assurer des résultats de découverte de service de haute qualité. La compréhension requise et partagée peut être fournie par l'utilisation des ontologies. Il y a plusieurs approches qui emploient des ontologies dans la découverte de service. Cependant, aucune d'elles ne considère l'utilisation de l'information contextuelle dans la manière de la découverte de service.
OWL est une ontologie qui peut être employée pour décrire sémantiquement des services. Elle permet des spécifications des services en termes de leurs *entrées, sorties, conditions*, qui doivent juger vrais avant l'exécution de service (appelée *les pré-conditions* en termes de OWL-S), et *post-conditions*, qui représentent l'état de l'environnement après l'exécution de service (appelée *effectives* en termes de OWL-S). C*OBRA* divise le monde en différents

domaines d'application. Chaque domaine est indiqué par sa propre ontologie qui fournit des concepts et des relations partagés pour la découverte de service. *OntoMat* [1] emploie des ontologies pour tracer les concepts employés par le demandeur de service aux concepts employés par le fournisseur de service. De cette façon, ces concepts peuvent être comparée et raisonnée.
CBSDP (*Context Based Service Discovery Protocol*) est un protocole de découverte de service pour les réseaux ads-hoc. CBSDP emploie des ontologies pour interpréter les données échangées pendant l'exécution de service.

Une partie du problème de la découverte des services est les problèmes du matching du service, où les descriptions des services publiés sont examinées avec les requêtes des utilisateurs afin de trouver un service ou un ensemble de services qui satisfait ces requêtes.

II.2.5. L'approche du Matching

II.2.5.1. Algorithme de Matching

Le mécanisme de recherche est basé sur cinq filtres. Ces filtres se basent sur différents aspects de correspondance entre requêtes et annonces :

1. **le contexte**. La requête et l'annonce doivent partager le même domaine. Le filtre calcule les distances entre les différents mots-clés d'annonce et de requête. Cette distance est basée sur les liens entre les mots tels que la généralisation, la spécialisation et l'association positive.
2. **le profil**. La requête et l'annonce doivent partager un certain nombre de mots dans leurs spécifications. Ce filtre compare les occurrences des mots de l'annonce et de la requête.
3. **la similarité**. La structure de la requête et la structure de l'annonce doivent être similaires, c.à.d. les concepts utilisés, par exemple dans la partie *input*. Au lieu de faire une comparaison de l'ensemble du document, on compare les différentes sections (input, output, pré condition, ...) avec le filtre de contexte.
4. **la signature**. La signature de l'annonce doit ressembler à la signature de la requête. On compare les différents termes (chaque paramètre individuellement) grâce au filtre de similarité. De plus, pour les paramètres *input* et *output*, on vérifie également que les types sont compatibles (équivalent, sousclasse, sur-classe).
5. **les contraintes**. L'utilisateur voulant utiliser un service doit vérifier les pré-conditions de ce service. De même, le service doit répondre aux attentes de l'utilisateur.

La composition de ces filtres permet d'établir différents degrés de correspondance :

- **exact match**. Le service correspond parfaitement à la requête.

On compare la requête avec l'annonce au moyen des cinq filtres.

- **plug in match**. Le service annoncé peut correspondre à la requête. Par exemple, l'utilisateur désire un service triant une liste d'entiers. Le médiateur propose un service de tri de liste d'entiers et de strings. On compare la requête avec l'annonce au moyen des deux derniers filtres : signature et contraintes.
- **relaxed match**. Cette relation détermine que la distance sémantique de l'annonce et la requête est inférieure à un seuil déterminé. On compare la requête avec l'annonce au moyen des trois premiers filtres. Il est donc le moins consommateur de ressource. En toute logique, les résultats du *relaxed match* devraient inclure les *exact* et *plug in match*.

Le ***matchmaker*** est un programme désigné pour accepter les requêtes et de donner une réponse pour ces requêtes en appliquant une technique de matching donnée.

Le service *direct matching* est lorsque le *matchmaker* examine une seule description de service, parcontre le service *aggregate matching* est lorsque le *matchmaker* examine les descriptions d'un groupe de services [59].

II.2.5.2. Matching des services web (*Services Web matching*)

Pour obtenir un résultat correct du matching, la sémantique des services, des utilisateurs et les domaines d'applications doit être capturée dans un format avec qui le *matchmaker* pourra comprendre cette sémantique. Ce dernier doit savoir comment utiliser chaque sémantique pour trouver le service approprié.
Le processus du service du matching est composé en quatre composants :

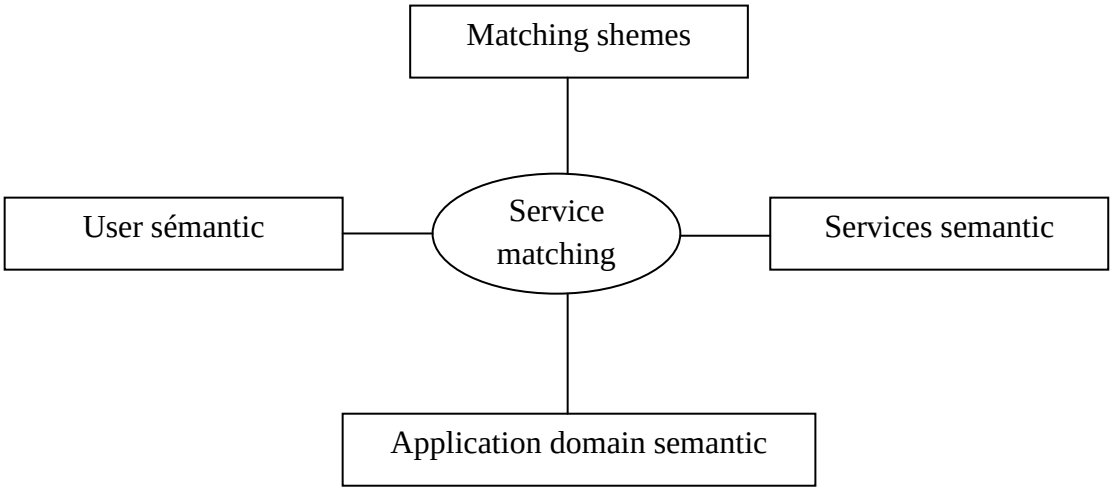


Figure II. 3. Composants of the service matching process

- **La sémantique des services (service semantic)**

Elle modélise la sémantique obtenue des descriptions des services web. Chaque sémantique contient les spécifications des services.
Le tableau suivant illustre une comparaison entre les services web conventionnels et les services web sémantiques.

L'aspect de Comparaison	Service web conventionnel	Service web sémantique
Service demandeur	Humain	Système/humain
Service de description	Text-based	Ontologie-based
Service du matching	Syntactically-based	Semantically-based
Service de découverte	Manuellement	Automatiquement
Service de composition	Manuellement	Automatiquement
Complexité	Simple	Complexe

Tableau II. 1. La comparaison entre le service web conventionnel et le service web sémantique

- **La sémantique des utilisateurs (user semantic)**

Le type spécifications demandées par l'utilisateur doit être compatible avec le type de spécifications capturées dans les descriptions du service.

- **La sémantique du domaine d'application (Application domain semantic)**

Les requêtes des utilisateurs peuvent ne pas être exactement similaires aux descriptions des services, d'où un processus de médiation est nécessaire.
Le facteur qui affecte ce processus est comment extraire la sémantique dans le domaine d'application. [14].

- **Le schéma du matching (Matching shemes)**

Il indique comment les services seront *matchés* en respectant un type donné de spécifications.
Il décrit comment la sémantique extraite dans les services, les utilisateurs et les domaines d'application, sera utilisée afin de déterminer un résultat correct.

II.3. La composition de services web

Les services web sont considérés comme des applications modulaires indépendantes, auto-descriptives, qui peuvent être publiées, localisées, et invoquées à travers le Web.

Si aucun service web ne peut satisfaire une requête d'un utilisateur, il devrait y avoir une possibilité de combiner un ensemble de services web existants pour accomplir cette requête.

D'après *Fensel* [25], les services peuvent interagir de manière intelligente en étant capables de se découvrir automatiquement, de négocier entre eux et de se composer en des services plus complexes.

II.3.1. Définition de la composition de services web

Composer les services entre eux, c'est entrelacer leurs séquences d'actions, de manière à obtenir des séquences qui correspondent aux séquences d'actions des besoins exprimés par le client dans sa spécification.

Nous pouvons définir la composition de services web comme étant l'assemblage, la coordination et la mise en place d'un ensemble de services afin d'obtenir un nouveau service dit composite capable de répondre à des requêtes cliente ne pouvant être satisfaite par l'un des Web services disponible.

II.3.2. Problèmes liés à la composition de services web

Lors de la composition des services web, les problèmes qui peuvent apparaître sont :

- Quelle est la description des services web (exemple de langage de description à utiliser) comme le WSDL, OWL-S, etc.
- Comment la base des services composants sera elle crée.
- Quelles sont les spécifications des services à enregistrer dans cette base de services.
- Comment doit être spécifiée la requête du client.
- Comment faire le matching entre les spécifications de la requête et les spécifications des services.
- Comment combiner les fonctionnalités des différents services pour construire le service composite.

II.3.3. Classification des services web selon leur modèle d'interaction et de composition

Les services web peuvent être classés selon leur modèle d'interaction comme suit :

II.3.3.1. Modèle de service web Atomique

Un service web atomique est décrit comme une boite noir, c'est-à-dire qu'il est spécifié en terme de ses entrées/sorties ainsi que des pré-conditions, sans se soucier du fonctionnement interne du service.

Le genre de composition associée à ce modèle d'interaction est une composition séquentielle,

II.3.3.2. Modèle de service web comportemental

Les services web basées sur un modèle comportemental sont souvent connus sous le nom de boite grise, ils sont décrits par l'ordre d'exécutions de leurs opérations.

Le genre de composition dans cette classe sera concurrentiel, c'est-à-dire qu'à un moment donné plusieurs services peuvent être actifs et s'exécutent en concurrence simultanément,

On distingue dans cette classe deux types de modèles d'interactions :

1. **Les services web basés sur l'exécution :** le comportement des services est décrit par une séquence linéaire d'actions, chacune décrivant une seule exécution. Par conséquent, dans chaque séquence, chaque point d'exécution a un seul point successeur.

2. **Les services web basés sur l'arbre :** chaque point d'exécution peut avoir plusieurs points successeurs.

II.3.4. Architecture d'un environnement de composition

La figure suivante illustre l'architecture générale d'un environnement de composition de service web.

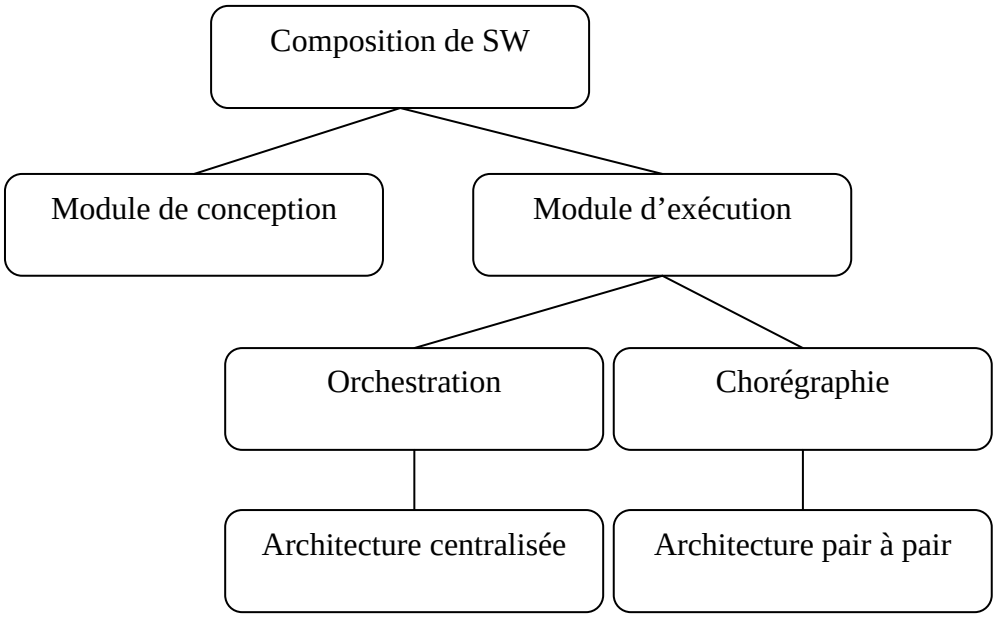


Figure II. 4 . Architecture générale d'un environnement de composition de services web

Du point de vue architectural, la composition des services web peut être divisée en deux modules :

1. **Le module de conception (La synthèse de composition**): consiste à spécifier comment coordonner les services web disponibles (trouver un plan de composition).

2. **Le module d'exécution (Orchestration)** : le problème d'orchestration traite la coordination de plusieurs composants du service web, et vérifie le contrôle du flux des données afin de garantir l'exécution correcte des services web composites,

- **L'orchestration** : Elle décrit les interactions entre les services web au niveau message, et l'ordre de leurs exécutions.

- **La chorégraphie** : Dépiste l'ordre des messages qui peuvent faire participer les participants.

Les deux architectures retenues pour l'orchestration et la chorégraphie sont :

- **Centralisé (Médiateur):** il y a un seul service nommé médiateur qui a un rôle spécialisé dans le contrôle d'opérations d'interaction des autres services, les autres services sont appelés les services composants.

- **Pair à pair (Peer to peer) :** dans ce cas, tous les services sont des services composants. Le contrôle d'opérations d'interaction entre les services est fait explicitement par chacun d'entre eux.

II.3.5. Présentation de quelques techniques de composition de services web

Nous décrivons dans ce qui suit quelques approches de composition en se basant sur le modèle d'interaction des services web participant à la composition.

II.3.5.1. Composition séquentielle des services web

Les méthodes de composition dans cette approche se situent soit dans les techniques du workflow ou bien la planification IA.

II.3.5.1.1. Composition de services web en utilisant les techniques du Workflow

Un Workflow est une spécification dont les participant dans un processus travaillent ensemble pour exécuter ce processus dés son début jusqu'à sa terminaison

II.3.5.1.2. Composition de services web en utilisant la planification IA

Plusieurs efforts de recherches ont été rapportés pour résoudre le problème de la composition des services web par l'intermédiaire de la planification IA (Intelligence Artificielle).

La planification, discipline de l'intelligence artificielle, cherche à concevoir des systèmes capables de générer automatiquement, grâce à une procédure formalisée, un résultat articulé, sous la forme d'un système intégré de décisions appelé plan-solution.

Quelques méthodes de composition de service web basées sur la planification IA :

1. Méthodes de composition basée sur des planificateurs

2. Méthodes de composition basée sur le matching

3. Composition de Web services en utilisant synthèse de programme (Program Synthesis)

II.3.5.2. Composition concurrentielle de services web

Plusieurs auteurs adoptent les machines d'états finis comme formalisme de base afin d'exploiter le comportement des services web.

II.3.5.2.1. Technique de composition basée sur Interface Automate

Dans cette approche, un service web est modélisé sous forme d'Interface automate qui est un modèle d'interface proche des automates d'état finis.

II.3.5.2.2. Une approche de composition basée sur les graphes

Cette technique adopte la description sémantique d'un service web avec le langage OWL-S. Elle utilise le formalisme interface automates pour modéliser le comportement du service web.

II.3.5.2.3. Approche de composition basée sur l'échange de messages

Cette technique de composition basée sur le comportement du service web où toutes les exécutions possibles sont représentées dans un arbre. Pour se faire, une extension de la spécification WSDL a était développée. Cette dernière représente le comportement d'un service web autant qu'un automate non déterministe (FSM), où les transitions représentent des messages échangés entre le client et le service.

II.3.6. Langages de composition de services web

Ils existent plusieurs langages de définition pour la composition de services web où le flux des processus et les liaisons entre les services sont connus a priori.

- **WSFL (Web Services Flow Language)**

WSFL (*Web Services Flow Language*) est un langage basé sur XML créé par HP (*Hewlett Packard*) permet d'intégrer la composition de services web comme un processus métier statique. Il propose deux styles de composition :

- Modèle de flux (*Flow Model*) : il s'agit de la description de la succession d'étapes afin d'obtenir l'objet attendu. Ce modèle est aussi appelé processus métier.
- Modèle global (*Global Model*) : il s'agit des interactions entre deux services, appelées aussi contrat.

Selon le type de composition, le document WSFL comporte comme élément premier le nom du type (*flowModel* ou *globalModel*). Les services web intervenant dans le processus sont connus par l'intermédiaire de l'élément *serviceProvider*. Le processus est constitué de différentes étapes appelées activités (*activity*). Pour chacune de ces activités, nous devons faire apparaître quel service la met en jeu (*performedBy*), quelle opération est permise (*operation*), et par quel type de port (*portType*). L'élément *controlLink* permet d'établir des conditions, des liens hiérarchiques entre les activités, etc. En d'autres termes, il spécifie la succession des étapes.

- **WSCI (Web Services Choreography Interface)**

WSCI est un langage XML de description d'interface qui décrit le flux de messages échangés par un service web qui interagit de manière ordonnée avec d'autres services.

- **WSCL (Web Services Conversation Language)**

WSCL propose de décrire à l'aide de documents XML les services web en mettant l'accent sur les conversations de ceux-ci, c'est-à-dire les messages échangés et leurs ordres.

- **XLANG**

Créé par Microsoft, le XLANG est une extension de WSDL. Elle fournit en même temps un modèle pour une orchestration des services et des contrats de collaboration entre celles-ci.

- **BPEL4WS (Business Process Execution Language for Web Services)**

BPEL4WS (*Business Process Execution Language for Web Services*) est un langage de composition proposé par IBM, Microsoft. Il correspond à la fusion des deux langages du Workflow WSFL et XLANG.

BPEL4WS est un langage dont la syntaxe est basée sur XML, il décrit des processus métiers, et permettant la coordination des services Web (décrits en WSDL) dans un flux de processus. En d'autres termes, il s'agit du moteur de l'orchestration donnant une représentation indépendante des interactions entre les partenaires.

Trois éléments permettent à BPEL4WS de gérer le flux de processus : les transactions, les partenaires et les espaces de stockages.

- Les transactions (*Exception handling and Transactions*) : les transactions sont principalement utiles dans le contexte de BPEL4WS pour gérer les erreurs et les appels d'autres services si le service appelé est indisponible ou défaillant, etc.
- Les partenaires (*Roles and Partners*) : ce sont les différents services web invoqués dans le processus. Ils ont chacun un rôle spécifique dans un processus donné.
- Les espaces de stockage (*Persistance and Containers*): ils permettent la transmission des données. Le flux de processus BPEL4WS permet que ces données soient consistantes à travers les messages échangés entre les services Web. Un message peut prendre les types suivants : l'appel (*invoke*), la réponse (*reply*) et l'attente (*receive*).

La structure des différentes activités peut être soit séquentielle soit parallèle. Si cette dernière est parallèle alors plusieurs services peuvent être invoqués en même temps.

II.3.7. Autour de la composition des services web

L'objectif de la composition de service est de créer de nouvelles fonctionnalités en combinant des fonctionnalités offertes par d'autres services existants, composés ou non en vue d'apporter une valeur ajoutée. Étant donnée une spécification de haut niveau des objectifs d'une tâche particulière, la composition de service implique la capacité de sélectionner, de composer et de faire interopérer des services existants.

Malgré tous les efforts fournis sur la composition de services web, cette tâche reste encore très complexe, cette complexité vient en général des sources suivantes:

- Le nombre de services web disponibles augmente, donc l'annuaire de publication devient volumineux ce qui alourdi la recherche de ces services.
- Les services web peuvent être créés et mis à jour continuellement
- Les services web peuvent être développés par des organisations différentes qui utilisent des modèles de conception et de description différents.

Contrairement aux *business processes* qui sont exécutés de manière prévisible et répétitive dans un environnement statique, les services web composés s'exécutent dans un environnement inconstant où le nombre de services disponibles évolue très rapidement. De

plus, la forte compétition engendrée par la multitude de fournisseurs de services oblige les entreprises à adapter leurs services pour mieux répondre aux besoins des clients et ce à moindre coût. Ces deux facteurs imposent des contraintes fortes sur les systèmes qui délivrent des services composés. En conséquence, les *business processes* qui décrivent des services composés devront intégrer de première vue ces contraintes en montrant des possibilités réelles d'adaptabilité à leur environnement.
Des résultats concernant cette problématique commencent seulement à émerger. Les travaux existants s'intéressent à une modélisation abstraite des services et à la définition d'un cadre formel pour les composer. Des travaux récents de la communauté web sémantique [29] commencent à explorer des approches combinant des outils d'annotation de services et de planification de manière à pouvoir composer automatiquement des services en vue d'atteindre des fonctionnalités prédéfinies. Ce type d'approche constitue une alternative aux langages procéduraux de type *BPEL4WS* en permettant de générer l'implantation d'un service composite à partir de spécifications déclaratives de son comportement. D'un autre côté, d'autres travaux s'intéressent à la définition d'un cadre formel permettant de mieux comprendre les relations entre les propriétés globales d'un service composite et les propriétés locales de ses composants. La motivation étant de développer des techniques de vérification et de synthèse (construction) des propriétés d'un service composite à partir des propriétés des de ses composants. Tous ces travaux tentent de ré-exploiter et d'étendre des techniques existantes, telles que les logiques temporelles, l'algèbre des processus, les réseaux de Petri, etc.

II.3.8. Conclusion

La découverte des services web est un aspect important dans les technologies orientées services web. Les services web sémantiques sont un effort pour fournir une description sémantique nécessaire des services web afin que l'automatisation de la découverte soit faite. La découverte des services web ne peut pas se baser toujours la description complète et détaillée de la sémantique, elle doit être complétée par une un processus de découverte automatique.
Nous avons l'intention d'effectuer une recherche d'une approche pour la découverte des services web sémantiques qui simplifie la description sémantique exigée et de venir avec un algorithme de matching efficace.
Dans le chapitre IV, nous proposons notre algorithme de matching des spécifications des services web et de la requête de l'utilisateur, qui se base sur la substitution des contraintes. Puis nous venons par une approche de composition des services où nous avons utilisé la notion d'agents qui servent à donner à notre approche une flexibilité au niveau d'ajout, suppression et mise à jour des services offerts.

CHAPITRE III

Etat de l'art sur le service du matching

Chapitre III

Etat de l'art sur le service du matching

III.1. introduction

La découverte des services web sémantiques nécessite d'avoir un *matchmaker* capable d'extraire les différentes sémantiques des différents composants (section 2, chapitre II) dans un format déterminé. Ces sémantiques extraites seront analysées et comparées selon un aspect de comparaison et un principe de *matching* spécifiés. Le *matchmaker* doit suivre différents schémas de *matching* pour déterminer le résultat correct et le degré de correspondance entre la description du service et celle de la requête du client.

Ce chapitre est consacré à un état de l'art sur une proposition d'un algorithme de matching et d'une approche dynamique de composition de services. Dans la première section, nous présentons les concepts de base sur le matching des services web, qui sera suivie d'une section où nous allons présenter un état de l'art qui définit les notions utilisées dans le matching sémantique des spécifications fonctionnelles d'un service web. Dans la dernière section, nous présentons quelques travaux liés à la problématique de notre travail.

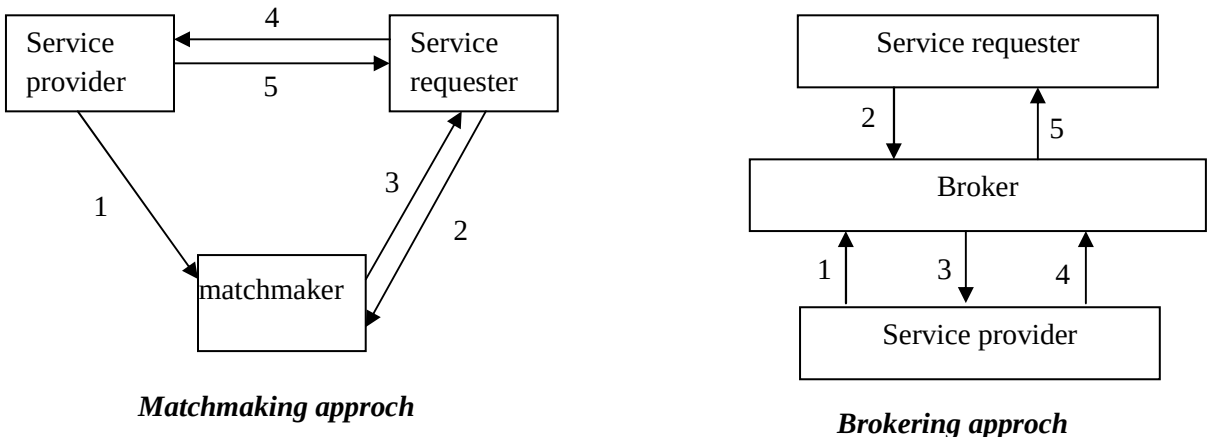
III.2. Le matching des services web sémantiques

L'étude des principes et concepts de base pour le matching des services web se résume dans les points suivants :

- L'étude entre *matchmaking* et *brokering approches* afin de répondre aux questions des utilisateurs.
- Introduction des ontologies qui sont adoptées dans les approches proposées.
- Description des approches utilisées dans la représentation des services et leurs limitations.

a) La différence entre le matchmaking et le brokering approches

L'architecture SOA (*Service Oriented Architecture*) définit des différents composants ; le service *requester* qui demande le service ; le service *provider* qui développe et publie les services et un service de découverte qui match les requêtes avec les services publiés afin répondre à ces requêtes.
Généralement, les deux approches suivantes, présentées dans la *figure suivante*, peuvent accomplir la tache attendue [21].



Matchmaking approch

1 : publier.
2 : soumettre la requête.
3 : retourner la liste des provider possibles.
4 : sélectionner un provider et soumettre la requête.
5 : retourner le résultat.

Brokering approch

1 : publier.
2 : soumettre la requête.
3 : sélectionner un provider et soumettre la requête.
4 : retourner le résultat.
5 : retourner le résultat.

Figure III. 1. L'approche matchmaking et l'approche brokering

Dans la première approche, le service *requester* soumet une requête au *matchmaker*, ce dernier cherche dans les services publiés, celui qui est le plus approprié qui répond à la requête en appliquant une technique de matching donnée. Ensuite, le *matchmaker* retourne le résultat au service *requester* qui prend en charge le contacte avec le service provider pour son exécution. Dans la deuxième approche, le service *requester* soumet la requête au *broker* qui cherche un ensemble de service qui peuvent répondre à la requête en appliquant une technique de matching, puis il sélectionne un service provider en respectant quelques caractères, généralement des caractéristiques de l'environnement de l'exécution comme la latence, l'équilibrage de charge, la bande passante…etc. puis il communique avec le service *provider* pour obtenir le résultat qu'il retourne au service *requester*.
La différence entre ces deux approches est que dans la deuxième, le service *provider* est caché pour le service *requester*, ainsi l'optimisation de l'environnement de l'exécution est obtenue par le *brokering* en gardant la trace de l'exécution.

b) Les ontologies

L'ontologie représente la partie de l'architecture du web sémantique, qui concerne le domaine de conceptualisation qui est une vue abstraite pour le domaine représenté. L'ontologie doit inclure les entités du domaine et les relations existantes. Elle doit dépasser quelques problèmes qui peuvent apparaître dans le processus de conceptualisation du domaine, par exemple : l'incompatibilité des définitions des domaines des attributs, l'incompatibilité du

niveau d'abstraction (une même entité est représentée dans des niveaux différents d'abstraction).
Lors de la création de l'ontologie, quelques caractéristiques doivent être prises en considération, à savoir : la *cohérence* (les déductions obtenues par les définitions doivent être consistantes), l'e*xtensibilité*(les définitions ajoutées ne doivent pas violer celles existées), la *minimalité* (les définitions ne doivent pas être redondantes)…
Il existe deux approches pouvant être adoptées dans le processus du matching : *consensus approch* [7] et *multi-ontology approch* [34]. Dans la première, chaque domaine d'application est décrit par une seule ontologie, alors que dans la deuxième, on peut avoir plusieurs ontologies différentes utilisées pour représenter un seul domaine d'application.

c) *Le service de représentation*

Les services ne sont pas des composants *middleware*, qui sont faciles à être décrits par des interfaces RPC [39], ils peuvent représenter des systèmes plus complexes tels que les *business processes* [16], d'où la nécessité des modèles plus avancés pour décrire les services web, qui peuvent être découverts et composés automatiquement. Il y a plusieurs architectures pour les représenter comme UDDI, OWL-S, WSMO. Chacun possède sa propre description de services.

1- UDDI (*universal description, discovery and integration*):

Son but principal est de trouver une représentation centralisées pour les services web, la description du service est conceptuellement divisée en trois types (*white page, yellow page et green page*). UDDI fournit différentes API pour la publication et information mais il ne fournit pas de la sémantique.

2- E-speak :

Il utilise la notion de ressource pour représenter le service omniprésent à travers le web. La ressource est décrite par un vocabulaire prédéfini par un service générateur fourni par l'architecture du E-speak. Le service *provider* enregistre les ressources dans un registre commun ; mais l'inconvénient de E-speak est qu'il y a des limitations dans l'implémentation.

3- LARKS (*Language for Advertisement and Request for Knowledge Sharing*):

C'est un langage utilisé pour la description des capacités des agents, comme un agent peut être vu comme un service, donc LARKS peut être utilisé pour la description des services. Il possède une description structurée qui inclue (*inputs, outputs, types, contexts, contraintes et fonctionnalités de l'agent*), le concept de l'agent est décrit par des *keyword* et ses fonctionnalités sont décrites par une description sous forme de texte. Le vocabulaire utilisé dans la description est défini par une ontologie nommé (*concept*) mais ne fournit pas un moyen pour décrire le comportement de l'agent.

4- EbXML

Il fournit une architecture pour *business electronic data interchange.* L'infrastructure de ebXML inclut :

- *The colaborateur protocol profile* qui définit la structure de données XML qui décrit le processus de communication, la sécurité demandée et les informations du contact.
- *Registry and repository* :chaque *registry* comprend BOV (*Business Operational View*) et FSV (*Functional Service View*).

Un service est principalement décrit par des ensembles différents de *business documents* (*UML diagrams* inclus dans BOV, *WSDL files* inclus dans FSV)

5- DAML-S (*Darpa Agent Language Services Ontology*) / OWL-S (*OWL-services-coalition*)

DAML-S est une ontologie pour la description des spécifications des services web basé sur le langage DAML+ OIL. La nouvelle version de DAML-S est connue par OWL-S (*Web*

Ontology Language). Elle décrit le service en trois composants : *service profile, service model et service grounding.*
La description par un texte n'est pas une approche convenable lorsque l'automatisation du service est demandée. Il est plus important d'utiliser les approches basées sur les *keywords.*

6- WSMO (*Services Web Modelling Ontology*):

L'ontologie WSMO est basée sur WSMF (*Service Web Modeling Framework*), elle se compose de : *ontologie, goal, médiators et services.*
Ontologie représente la définition du concept, la définition des relations, les propriétés non-fonctionnelles…*goal* comporte les propriétés non fonctionnelles, les médiateurs utilisés, les post-conditions…etc. Un *mediator* est utilisé pour lier les composants hétérogènes et définit le mapping et les transformations entre ces éléments. Mais WSMO manque de la représentation des comportements des services web.

d) Service direct matching

<table>
<tr><td rowspan="7">Le service de matching direct</td><td rowspan="2">Non structure</td><td>Syntaxique</td><td></td><td></td></tr>
<tr><td>Sémantique</td><td></td><td></td></tr>
<tr><td rowspan="5">Structuré</td><td>Non fonctionnel</td><td></td><td></td></tr>
<tr><td rowspan="4">Fonctionnel</td><td rowspan="2">Haut niveau</td><td>Sémantique</td></tr>
<tr><td>Syntaxique</td></tr>
<tr><td rowspan="2">Bas niveau</td><td>Sémantique</td></tr>
<tr><td>Syntaxique</td></tr>
</table>

Tableau III. 1. Classification du Service Direct Matching

Les approches de *services direct matching* sont soit *structurée (structure-based matching)*, qui différencient entre différents types de spécifications durant le processus du matching ou *non-structurée (non-structure-based matching)* qui ne font pas cette tache, d'où l'utilisation des techniques de matching à base de mots clés (*keyword*).
Les *structure-based matching approches* peuvent être *fonctionnelles* qui utilisent les spécifications fonctionnelles comme *goals, roles, context et behavior*, ou *non-fonctionnelles*, qui utilisent les spécifications non fonctionnelles. Actuellement ces dernières sont focalisées sur les paramètres de la qualité de service (QoS).

e) Service de composition et d'agrégation

Un service de composition est un service virtuel qui consiste à collaborer deux ou plusieurs services afin d'accomplir une tache donnée. Il peut être décomposé en deux phases :

- *High-level Aggregation phase* : le service composé est examiné par rapport aux aspects fonctionnels demandés par l'utilisateur comme le but, le context et le comportement.
- *Low-level execution phase* : les détails de l'exécution sont examinés afin d'assurer les propriétés ACID (*atomicité, cohérence, isolation, durabilité*), la coordination, le contrôle des compositions de l'exécution.

Le service composition peut être statique (les composants utilisés sont pris manuellement) ou dynamique où les composants sont identifiés au moment de l'exécution. L'inconvénient du premier type est qu'il manque de représentation sémantique car la plus part des approches se base principalement sur le model WSDL pour décrire les messages et les données.

Le matching des services composés	Stucturé	Fonctionnel	Haut niveau	Sémantique
				Syntaxique
			Bas niveau	Sémantique
				syntaxique

Tableau III. 2. Classification des techniques du matching des services composés

III.3. Etat de l'art sur la proposition

Le schéma du matching fournit le concept et les bases qui constituent le processus de *matching*. L'obstacle majeur pour atteindre pleinement le service *matching* est l'intervention de l'être humain afin de déterminer le résultat correcte du *matching*. Il doit donc être capable de déterminer le résultat correct en respectant les buts de l'utilisateur. Par conséquent, la sémantique doit être capturée dans un format où le *matchmaker* pourra la comprendre [59].

Pour automatiser le processus du matching [59], le schéma du matching doit être capable de :

- Séparer entre les présentations des services (créées pour la publication) et leurs descriptions (créées pour l'automatisation de processus du matching).
- Surmonter l'interopérabilité sémantique, capturer les rôles supportés par les services et ceux joués par l'utilisateur, car les services sont désignés pour servir une catégorie d'utilisateur.
- Capturer les buts des utilisateurs et des services, car les services sont invoqués afin d'atteindre des buts prédéfinis, et cela pour vérifier si le service retourné atteint les buts désirés.
- S'adapter au comportement des services web pour faciliter la composition et l'agrégation des services web et supporter les exigences non fonctionnelles et non techniques.

III.3.1. Les type de spécifications

A partir des caractéristiques mentionnées ci-dessus, nous identifions les spécifications suivantes qui seront capturées dans les descriptions de services et les requêtes des utilisateurs ;

- **Les spécifications fonctionnelles de haut niveau (*High-level functional specifications*) :**

Elles concernent la description des fonctionnalités de haut niveau, par exemple : elles doivent inclure les informations sur les buts des services, les rôles, le comportement extérieur etc,

- **Les spécifications fonctionnelles de bas niveau (*low-level functional specifications*) :**

Elles concernent la description des fonctionnalités de bas niveau, par exemple : elles doivent inclure les informations sur les interfaces d'accès, les protocoles de messagerie etc,

- **Les spécifications non fonctionnelles (*Non-functional specifications*):**

Elles concernent la description des aspects non fonctionnels, par exemple les informations sur la qualité de service et sécurité.

- **Les spécifications non techniques (*Non-technical specifications*):**

Elles concernent la description des aspects de marketing par exemple les informations sur le prix, la remise, etc ;

La classification de ces spécifications est comme suit :

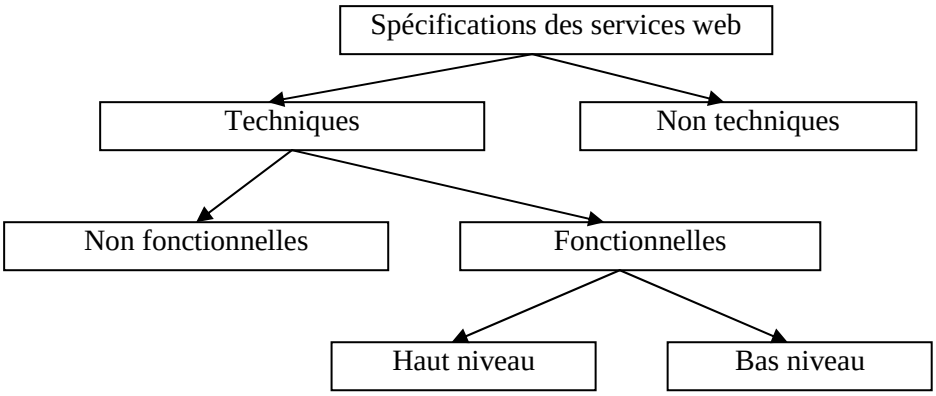


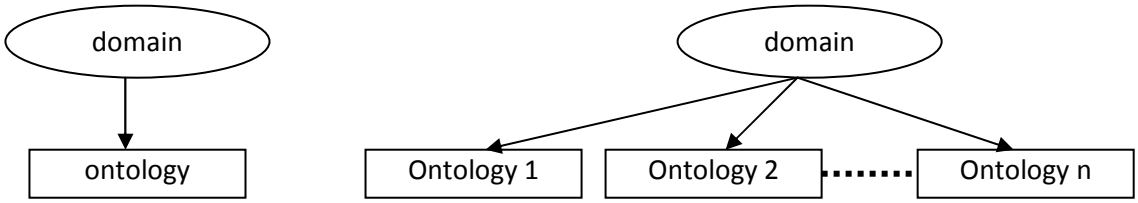
Figure III. 2. La classification des spécifications des services web

Le *matching* doit être fait à chaque étape, sachant où à chaque étape un seul type de spécification est analysé. Ce processus du *matching* doit commencer par les spécifications fonctionnelles de haut niveau qui fournit au *matchmaker* les informations pour atteindre les résultats corrects et ensuite le *matching* des autres spécifications permet de raffiner les résultats obtenus.

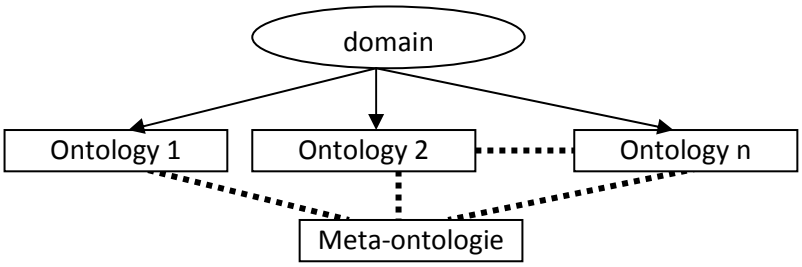
SWSMF (*Semantic Web Services Matching Framework*) est un cardre proposé par ELGEDAWY et al. [21,57] dont l'objectif est de :

a. Séparer la sémantique des services web de leur présentation, SWSMF possède deux vues, SAV (*Service Active View*) qui contient une machine qui permet de comprendre les informations pour automatiser le processus du matching, et SPV (*Service Passive View*) qui contient les informations qui ne sont pas dans cette machine comme les diagrammes UML, les documents d'échange...

b. Surmonter le problème de l'interopérabilité sémantique. SWSMF utilise le domaine des ontologies et combine entre l'approche de consensus et l'approche multi-ontologies, par la proposition de la nouvelle approche *'l'approche multi-ontologies'*.



L'approche de consessus L'approche multi ontologies



L'approche méta-ontologies

Figure III. 3. Les différentes approches d'ontologies

Dans l'approche proposée, un domaine d'application peut être représenté par plusieurs ontologies, à condition que ces dernières adoptent la méta-ontologie. Cette méta-ontologie capture les concepts et les opérations des domaines d'application. En adoptant cette méta-ontologie, la flexibilité de l'approche multi ontologie est maintenue et le processus du *mapping* devient simple, cela du fait que les ontologies ont la même structure et capturent le même type de la sémantique.

c. Pour capturer les buts des services web, SWSMF utilise SAVs en employant le modèle G+. Le modèle G+ permet d'établir le lien entre les objectifs, les contextes et scénarios pour leurs réalisations [59]. SWSMF permet au service web d'avoir plusieurs buts.

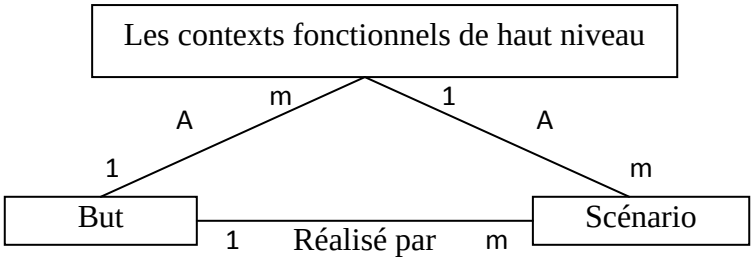


Figure III. 4. Le model conceptuel du G+

d. SWSMF utilise le concept de scénarios pour capturer le comportement extérieur des services web. Un scénario indique les interactions entre les utilisateurs et un service web afin d'aboutir au but de service. Plusieurs scénarios peuvent être utilisés pour un seul but et qui peuvent être représentés dans différents niveaux d'abstractions. SWSMF propose le concept du réseau de scenarios pour définir ces derniers. Ce concept est une partie du model G+.

e. SWSMF utilise le modèle G+ comme une base du service de composition car parmi les caractéristiques des buts, ils peuvent être composés pour former un but plus complexe.

f. SWSMF capture les rôles des services dans son SAV (*Service Active View*) et les rôles des utilisateurs dans UAV (*User Active View*). La méta-ontologie capture les rôles comme des concepts représentant les acteurs du domaine d'application. SWSMF n'exige pas que les rôles des services et utilisateurs doivent avoir le même concept.

III.3.2. La méta-ontologie pour modéliser le domaine d'application

La méta-ontologie proposée dans [22] spécifie les éléments et la sémantique capturés dans les ontologies des domaines d'application pour modéliser et matcher les spécifications fonctionnelles de haut niveau. Elle consiste en deux couches, la couche schématique (*schematic layer*) et la couche sémantique (*semantic layer*).

Couche sémantique	Les sémantiques fonctionnelles de substitution	
Couche schématique	Concepts	Opérations

Tableau III. 3. Les couches de la méta-ontologie

III.3.2.1. La couche *schématique*

Les éléments du domaine d'application (concepts et opérations) capturés dans la couche *schématique*, sont les éléments de base pour décrire les spécifications fonctionnelles de haut niveau des services web. Cette couche est responsable de définir ces éléments qui sont définis

par un ensemble d'attributs. Chaque attribut contient le nom, une description sous forme de texte et le rang qui doit être défini dans l'ontologie adoptée.

Nous extrayons les informations concernant les éléments du domaine d'application en utilisant les contraintes, nous définissons quelques principes sur les contraintes [59]:

Définition 1 : (contrainte comparative)

Une contrainte comparative sur un attribut $\boldsymbol{attr_k}$ d'un élément $\boldsymbol{E_i}$ est définie par ($\boldsymbol{E_i}$. $\boldsymbol{attr_k}$ λ r) tel que λ ε {=,≠,<,>,≤,≥}, et r est une valeur atomique de même type que $\boldsymbol{attr_k}$.

Définition 2 :(domaine de contrainte)

Chaque contrainte a son domaine correspondant qui indique l'attribut et l'élément auxquels la contrainte est restreinte.

Soit *const* une contrainte, associant $\boldsymbol{attr_k}$ à $\boldsymbol{E_i}$. Le domaine de *const* (dénoté : Ξ(const)) est $\boldsymbol{E_i}$. $\boldsymbol{attr_k}$.

Le domaine d'un ensemble de contraintes est formé de l'ensemble des domaines de l'ensemble des éléments correspondants.

Définition 3 :(domaine de l'ensemble de contrainte)

Soit l'ensemble de contraintes f_i = {*const1, const2…*}, le domaine de f_i (dénoté : Ξ(f_i)) est { Ξ(const1), Ξ(const2),…}.

III.3.2.2. La couche *sémantique*

La couche sémantique '*semantic layer*' capture la sémantique du domaine d'application utilisé dans le processus de médiation dans le processus du matching. Ces sémantiques sont capturées par les différents types de relations spécifiques au schéma du matching adopté. La couche sémantique proposée capture les sémantiques de substitution fonctionnelle.

La substitution fonctionnelle entre les attributs des concepts doit être prise en considération, les attributs du même nom doivent avoir différentes sémantiques. Ces sémantiques doivent être capturées dans le niveau du domaine des attributs du concept indiquant le mapping entre les valeurs de domaine de substitution. La sémantique de substitution doit être définie en respectant chaque opération définie dans le domaine d'application concerné. Elle doit être aussi capturée dans une manière basée sur le context. Le graphe de substitution de concepts (*concepts substitutability graph*) est proposé pour capturer les sémantiques de substitution des domaines basées sur les contextes.

III.3.3. Le graphe de substitution

Il est constitué de segments où chaque segment correspond à une opération du domaine d'application. Un segment du graphe de substitution est un graphe orienté, et montre la direction de la substitution. L'existence de lien du domaine de la source au domaine de destination signifie que le premier est fonctionnellement équivalent au dernier suivant la sémantique des opérations du segment. Il y a des conditions qui doivent être satisfaites d'abord par le contexte de l'exécution du but de l'utilisateur. En conséquence, chaque nœud du graphe possède un ensemble de contraintes de substitution, des fonctions de conversion correspondantes et une matrice de mapping des opérateurs.

III.3.4. Définition de quelques concepts sur la substitution des contraintes

a) Une contrainte de substitution

Une contrainte de substitution est une contrainte qui doit être satisfaite lors de la substitution afin de pouvoir remplacer le domaine de la destination avec le domaine de la source.

b) La fonction de conversion

La fonction de conversion est la fonction du *mapping*, qui convertie les valeurs d'un ensemble de départ aux valeurs d'un ensemble de destination. Elle prend une valeur d'entrée et retourne un ensemble de valeurs de sortie.

c) La matrice de mapping des opérateurs (OMM)

La matrice de mapping des opérateurs est une matrice qui montre le *mapping* entre les opérateurs des contraintes. il y a huit opérateurs qui sont utilisés: =, ≠, <,>, ≤, ≥,∈,⊆, d'où la matrice doit avoir huit lignes et huit colonnes, un élément de la matrice est égale à 1 lorsqu'il y a un mapping entre un opérateur de la source à celui de la colonne correspondante et 0 le cas contraire.

III.3.5. Le graphe de substitution des concepts ou *Concept Substitutability Graph*

Un graphe de substitution des concepts (*Concept Substitutability Graph* ou CSG) est un graphe multi-segment, tel que CSG=$\bigcup_{i=1}^{n}\{\langle op_i, V_C, E_C\rangle\}$, tel que $\cup$ est l'opérateur de l'union, $\langle op_i, V_C, E_C\rangle$ est le segment du graphe de substitution pour l'opération Op_i , n est le nombre des opérations dans le domaine, V_C est l'ensemble des nœuds du graphe, sachant que $V_C \subseteq \Delta_C \otimes \Delta_A$, tel que Δ_C est l'ensemble de tout les concepts et Δ_A est l'ensemble des attributs et $\otimes$ l'opérateur produit, E_C représente l'ensemble des bouts du graphe. Sachant que $E_{(a,b)} \in E_C$, $E_{(a,b)} = < V_a , V_b, \Pi_{(a,b)}, \Psi_{(a,b)}, \Omega_{(a,b)} >$ tel que $\Pi_{(a,b)}$ est l'ensemble des contraintes de substitution, et $\Psi_{(a,b)}$ est la fonction de conversion et $\Omega_{(a,b)}$ est la matrice de mapping des opérateurs.

III.3.5.1. Le modèle de but G+

Le modèle G+ [59] fournit le moyen pour décrire un but, son exécution et les capacités du service correspondantes (caractéristiques fonctionnelles) adoptés pour la méta-ontologie proposée.

Ce modèle est une extension de l'approche goal-scénario utilisée pour la modélisation de l'objectif afin de capter les spécifications de haut niveau des objectifs des services web.

L'état réel d'un domaine d'application à un instant donné peut être décrit par un ensemble de facteurs, ce que nous décrivons par un ensemble de contraintes. L'exécution d'un but implique le changement de l'état réel. Chaque changement peut être capturé par la manipulation des contraintes, commençant par celles de l'ensemble de concepts initial avant l'exécution de but (**pré-contraintes**), vers celles de l'ensemble de concepts final (**post-contraintes**) en passant par celles des ensembles des concepts qui apparaissent durant l'exécution(**describing-contraintes**).

Définition 4 : le modèle G+ est défini dans le format suivant : < *Op, Ctxt, SN* > sachant que *Op* est une opération représentant G, *Ctxt* est l'exécution des contextes définit par < *Pre, Desc, Post* > qui sont dénotés respectivement **pré-contraintes (**$Ctxt^{Pre}$**), describing-contraintes (**$Ctxt^{Desc}$**), post-contraintes (**$Ctxt^{Post}$**).** SN est le scénario utilisé pour l'exécution du G.

G+ adopte la notion de scénario pour décrire les étapes de l'exécution, un scénario montre un seul historique et il est modélisé par une séquence d'opérations.

Durant le matching des contraintes, la gestion du matching doit être de la partie qui garantie la satisfaction de ses contraintes (source) vers celle qui exige leurs satisfactions (destination).

Ensemble de contrainte	Source	Destination
Pre-contraintes	Utilisateur	service
Post-contraintes	service	Utilisateur
Describing-contraintes	service	Utilisateur

Tableau III. 4. Le matching de HLFC

Le *matchmaker* est responsable d'analyser tous les scénarios possibles fournis par le provider et l'utilisateur. Un chemin d'un nœud but à un autre nœud d'une opération (non-successeur) représente un chemin pour accomplir le but, il s'appelle *Goal Achievement Pattern* (GAP. Modèle de réalisation de but). Un GAP fournit un instant global de la façon que le but peut être accompli, il fournit toutes les informations pour le *matchmaker* afin d'extraire le comportement externe du service dans un contexte donné. En ayant les GAPs pour un réseau de scénarios implique tracer tous les chemins.

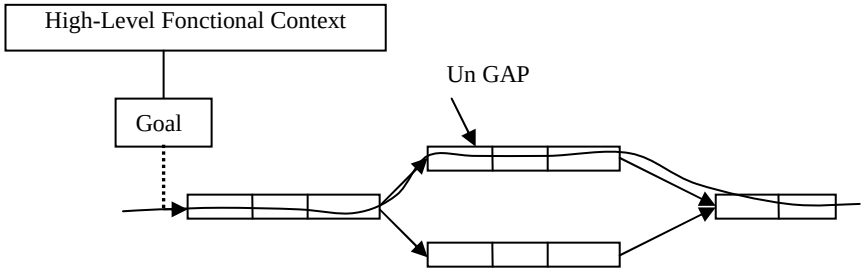


Figure III. 5. Un exemple d'un chemin GAP

Pour chaque réseau de scénarios, un ensemble de branches est visité. Pour visiter une branche, un ensemble de contraintes doivent être satisfaites. Cet ensemble de contraintes crée de nouvelles contraintes (un sous-contexte, sub-context) pour ce modèle de réalisation de l'objectif (GAP) qui sera additionné aux pré-contraintes du modèle G+. L'ensemble de toutes les contraintes du chemin de réalisation de but (d'une branche du réseau de scénarios) forment le contexte fonctionnel de Haut Niveau de ce GAP.

***Définition 5:* (GAP)**

Soit le but $G=\langle Op, Ctxt, SN \rangle$, le modèle d'accomplissement du but G est défini comme suit : $Gp=\langle Op, GCtxt, OpSeq \rangle$, tel que GCtxt = $\langle$ $Ctxt^{pre} \wedge SubCtxt$, $Ctxt^{Desc}$, $Ctxt^{Post}$ $\rangle$, OpSeq est la séquence d'opérations résultantes de la description de SN.

III.3.5.2. La justesse du GAP ou *GAP correctness*

La séquence d'opérations pour un GAP donné peut être vu comme une fonction de transformation σ de pré-contraintes du GAP aux post-contraintes par un ensemble de points de transitions sachant que chaque points de transition *x* possède un ensemble de contraintes f_x.

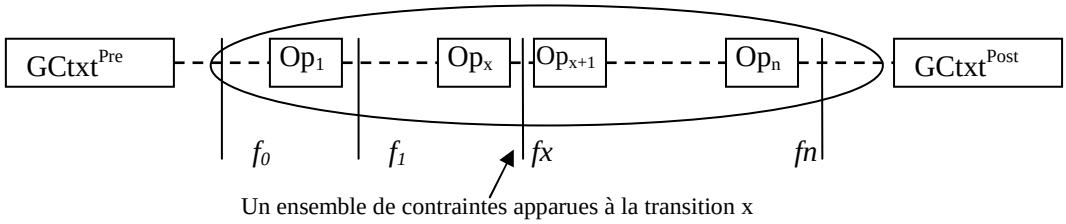


Figure III. 6. La fonction de la transformation de GAP

Pour invoquer une opération, ses pré-contraintes doivent être satisfaites. Un GAP est considéré correctement défini si $GCtxt^{pre}$ est transformé en $GCtxt^{post}$ par σ et chaque ensemble de contraintes à une transition satisfait les pré-contraintes de l'opération suivante.

***Définition 6* : La satisfaction de contraintes**

Soient $Const_i$ et $Const_j$ deux contraintes. $Const_i$ satisfait $Const_j$ (noté $Const_i \models Const_j$) si ($\Xi(const_i) = \Xi(const_j)$) $\wedge$ $const_i \Rightarrow const_j$

La détermination de la satisfaction des contraintes est très importante dans le processus du matching.

La satisfaction entre deux ensembles de contraintes (*constraint-sets satisfiability*) est plus complexe car il peut exister différents types de relations entre les éléments d'où on suppose qu'un ensemble de contraintes est satisfait lorsque chacune des contraintes de cet ensemble est satisfaite.

***Définition 7* : La différence sémantique**

Soient deux ensembles de contraintes f_i et f_j, tel que f_i et f_j appartiennent au même GAP, la différence sémantique entre f_i et f_j (dénoté $f_i \lozenge f_j$) est f_k, sachant que f_k est le plus grand sous ensemble de f_i tel que $(f_i - f_k) \models f_j$.

Alors dans cette section, nous avons présenté un état de l'art sur les notions et concepts de bases et le framework du matching sur lesquels s'appuie notre proposition. Nous avons l'idée de faire une extension et une amélioration pour les travaux réalisés par A. Tari et al. [59] en réalisant un algorithme de matching des spécifications fonctionnelles de haut niveau, qui représentent la sémantique des services à travers des contraintes en se basant sur le modèle G+ qui décrit les besoins et les effets de la réalisation d'un objectif à travers trois ensemble de contraintes: pré-constraintes, post-constraintes et les contraintes descriptives. Cet algorithme

sera utilisé dans une nouvelle approche de composition de services que nous allons définir ensuite. Des détails (définitions) et des théorèmes démontrés utilisés dans notre proposition seront expliqués dans le chapitre suivant.

Avant de détailler nos algorithmes, nous allons consacrer la section suivante pour discuter quelques travaux reliés à notre problème et ceux qui sont réaliser dans l'approche de la composition.

III.4. Quelques travaux reliés au problème du matching

Dans cette section, nous présentons quelques travaux réalisés dans le domaine de la découverte sémantique des services web, plus précisément dans le problème du matching d'un seul service ou le matching d'un ensemble de services pour une description d'une requête d'utilisateur ; appelé problème de la composition de service.

Nakala et Tari [61] proposent dans cet article, deux approches du service de découverte qui localisent les services qui sont conforment aux contraintes globales indépendantes. Dans la première approche, les services sont indexés par des valeurs qui sont attribués pour ses attributs restreints (les attributs restreints pour une contrainte spécifiée). Les services qui assignent des valeurs conformes à ces attributs sont combinés pour former un service composé. La deuxième approche utilise le domaine des règles afin de déduire les valeurs des attributs.

Les auteurs considèrent deux types de contraintes, le premier type (les contraintes locales) restreint les valeurs pour des attributs particuliers d'un seul service, tandis que le deuxième type (les contraintes globales) restreint les valeurs de deux ou plusieurs attributs de plusieurs services composants. La classification des contraintes globales revient à la complexité de leur résolution et cela étant des contraintes strictement dépendantes et non strictement dépendantes. Ils proposent une technique de matching sémantique pour localiser les services qui satisfont les contraintes globales indépendantes. Une structure de donnée bidimensionnelle est définie appelée « *slot list* » pour retrouver les services, les services sont indexés à la base des valeurs assignées à ses attributs restreints. Cet indexe sert à retrouver rapidement les services. Les tuples des valeurs conformes sont déterminées et les services qui assignent ces valeurs sont retournés afin de les composer.

Nakala et Tari utilise la méta ontologie proposée par Elgedawi [22]. La fonctionnalité du service est décrite par trois éléments : l'objectif, les états de transition et les transformations, les états sont décrits par les valeurs qui peuvent être assignées aux attributs, les transformations sont décrites par les préconditions et les postconditions, les conditions définissent les relations entre les attributs et les valeurs. La deuxième approche du matching proposée, utilise les règles pour trouver les valeurs conformes à la contrainte globale indépendante.

Ils définissent le type du service comme triplet [O,C,R], O est une opération, C est le concept et R est le rôle. L'approche proposée utilise la technique d'optimisation locale et la technique à base de la dérivation. Ils définissent un ensemble de services [s_1, …,s_m] sont conformes à *gc*

tel que *gc* est une contrainte qui restreint les attribut $[a_1, \ldots, a_n]$ des services de type $[S_1, \ldots, S_m]$ si :

1. $\forall s_i$, s_i est du type S_i
2. $\forall s_i$, s_i est décrit utilisant un attribut a'_i tel que $a'_i \in \{ \}$ et
3. $[v_1, \ldots, v_n]$ assignés à $[a'_1, \ldots, a'_n]$ de $[s_x, \ldots, s_y]$ conforme a *gc*.

$[a'_1, \ldots, a'_n]$ de $[s_x, \ldots, s_y]$ sont les attributs restreints.

L'approche proposée consiste en trois phases :

La première phase localise les services (services candidats) composants, la deuxième phase identifie les attributs restreints des services candidats (phase d'indexation), tandis que la troisième retourne les valeurs conformes et combine leurs services pour les composer. Cette approche génère dans la deuxième phase un ensemble de listes appelées *Slot lists* pour une contrainte globale donnée. Elle génère pour chaque attribut une liste. Un service est inclus dans une liste si on peut affecter une valeur pour l'attribut pour lequel la liste est générée.

Cette approche modélise la contrainte globale sous forme de graphe, tel que les nœuds représentent les attributs et les arcs représentent les relations (comparaison binaire) entre les attributs. Les valeurs conformes des attributs (valeurs consistante de l'arc) sont identifiées par la génération des instances pour le graphe.

Le calcul des instances consistantes d'un arc reste un problème NP-difficile (d'une complexité exponentielle). L'approche propose deux techniques de calcul, la première emploie une technique optimisée avec l'algorithme de recherche avec retour arrière appelée *optimised approach*, la seconde les déduit dans les règle du domaine appelée *dérivation-based approach*.

Optimised approach essaye de générer une instance d'arc consistant pour chaque attribut. Premièrement, une valeur est sélectionnée et attribuée pour un attribut, puis, en se basant sur cette valeur, les valeurs de l'arc consistant sont sélectionnées pour les attributs restants. Si une valeur ne peut pas être attribuée, la technique procède à la valeur suivante sinon elle va effectuer un backtrack. La complexité en temps d'exécution de cet algorithme est exponentielle. Si une contrainte restreint *p* attributs, et il existe *m* tuples de valeurs et chaque slot contient *q* services, la complexité de la composition est $m(p^q)$.

Derivation-based approach utilise le domaine des règles pour dériver les instances de l'arc consistant. Il est utilisé aussi pour faire une approximation pour la contrainte globale indépendante à une contrainte globale strictement dépendante. Cette approximation reste toujours parmi les problèmes NP-difficiles.

Les deux techniques proposées montrent une complexité inferieure à celles des techniques existantes, elles localisent les services composants dans un temps polynomial, l'inconvénient est qu'elle retourne des entrées dupliquées. Si *n* valeurs conformes peuvent être assignées à un ensemble d'attributs, *n-1* entrées dupliquées sont retournées.

Tari a proposé dans [59] une technique de matching des spécifications de bas niveau utilisées comme un filtre des résultats de matching des spécifications fonctionnelles de haut niveau en se basant sur les interfaces. Il définit le point d'accès d'un service comme étant l'interface qui contient des méthodes indivisibles offrant un accès uniforme à l'objet service qui fournit le service, il a défini les types de base et complexes incluant les méthodes, les propriétés et exceptions, interfaces etc… Une interface est une forme spéciale d'un tuple, une instance de T_f possède la structure $\langle \alpha_1, \ldots, \alpha_n \rangle$ où α_i est une instance d'un type, d'une méthode ou d'une propriété. Il a défini des règles d'équivalence entre les types structure afin d'assurer la conformité. En plus, il a proposé une approche de composition de haut niveau des services web. Cette approche est une approche séquentielle de telle sorte une solution au problème d'agrégation est une séquence de services web qui remplace fonctionnellement le chemin GAP (Goal Achievement Pattern) requis sans interaction avec les utilisateurs, il utilise la notion de *Plug-in GAP* et de *Computation chunk* qui est défini comme (*ServiceID, Start, End*) où *Start* et *End* sont respectivement les indices du premier et dernier état du comportement requis du *plug-in* GAP du service.

L'algorithme de composition proposé consiste à obtenir une liste de plug-in GAPs d'un GAP requis en parcourant le service de registre UDDI pour rechercher les services satisfaisant les parties continues du GAP requis, puis il vérifie l'existence qu'une séquence de ces chunks qui matche avec toute la requête. Pour cela il utilise une structure de données qui organise la liste des chunks en un tableau de piles. Le résultat du processus du matching est un tableau de piles correspondant au modèle de comportement requis. Le processus de la composition commence par vérifier les chunks dans la pile correspondante au premier état cible S_0, noté Pile-0. Pour chaque chunk dans la Pile-0, on crée une nouvelle pile contenant tous les chunks qui commencent avec la valeur de son champ End puis il examine les piles descendantes par un appel récursif pour voir si un de leurs chunks couvre la sous séquence cible restante en vérifiant si son champ End contient l'indice du dernier état cible.

La figure suivante illustre ce processus de composition.

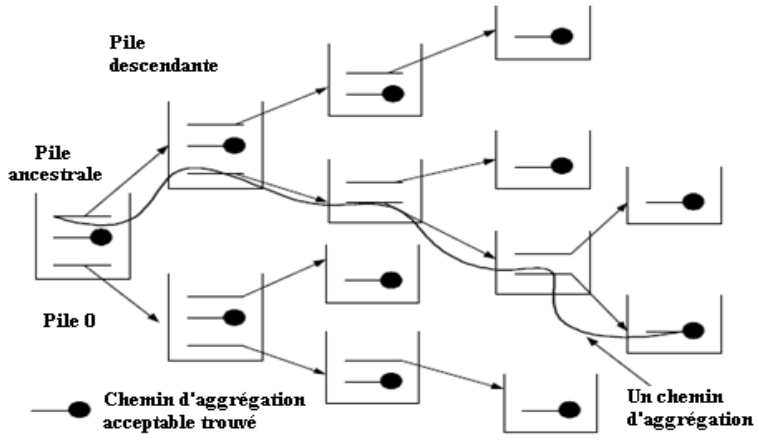


Figure III. 7. GAP Aggregation Approach

Cette approche montre de bons résultats par rapport aux approches de composition concurrentielles, mais elle reste lourde dû à la complexité de l'algorithme de composition ainsi au niveau des structures de données utilisées qui nécessitent un grand espace mémoire.

Tari a proposé une approche de composition de bas niveau pour les services web, il considère les services comme des partie indivisibles. Pour le client, ces services sont les méthodes ainsi que les éléments de données dans les structures de données impliquées. Il définit un ensemble d'opérateurs pour réaliser la composition et de combiner les différents éléments des interfaces des services. Afin de réaliser la composition et définir comment les composants interagissent, l'approche consiste à insérer des éléments de données dans les méthodes, en créant de nouvelles signatures de méthodes qui incluent de nouvelles listes de paramètres basées sur la composition des éléments. Les éléments qui sont composés dans une méthode doivent être dans le même domaine de la méthode. Pour cela, il utilise l'opération union avec une modification de la signature.

Cette approche a été proposée pour renforcer la découverte sémantique des services en intégrant les spécifications fonctionnelles de bas niveau d'abstraction. Le bas niveau d'abstraction permet la capture des informations sur les accès aux interfaces, de la bande passante ainsi que sur les protocoles de communication. Cette approche permet la combinaison de plusieurs services pour la création de services composites (virtuels) en se basant sur la conformité entre les types services.

Dans [45], naylor et al. proposent une approche de matching dans le contexte de la sémantique mathématique. Ils discutent le problème de matching des services mathématiques où la sémantique joue un rôle dans la détermination de l'applicabilité des services. La plus part des travaux réalisés se focalisent dans la description des services sous forme de texte structuré ou non structuré, ce qui nécessite simplement une recherche par mots clés ou la résolution par des contraintes pour effectuer le matching.

Les auteurs utilisent les descriptions de OpenMath pour les préconditions et postconditions et proposent une architecture qui est constituée des composants suivants:

- Une interface client qui est utilisée par l'utilisateur pour spécifier sa requête ;
- Un matchmaker qui contient le moteur d'inférence et le module du matching ;
- Les algorithmes du matching qui définissent la logique du matching ;
- Les ontologies mathématiques telles que les composants de OpenMath ;
- L'enregistrement des services où les descriptions du service mathématique sont sauvegardées
- Les services web mathématiques qui sont accessibles par le web.

L'architecture proposée est définie dans la figure suivante.

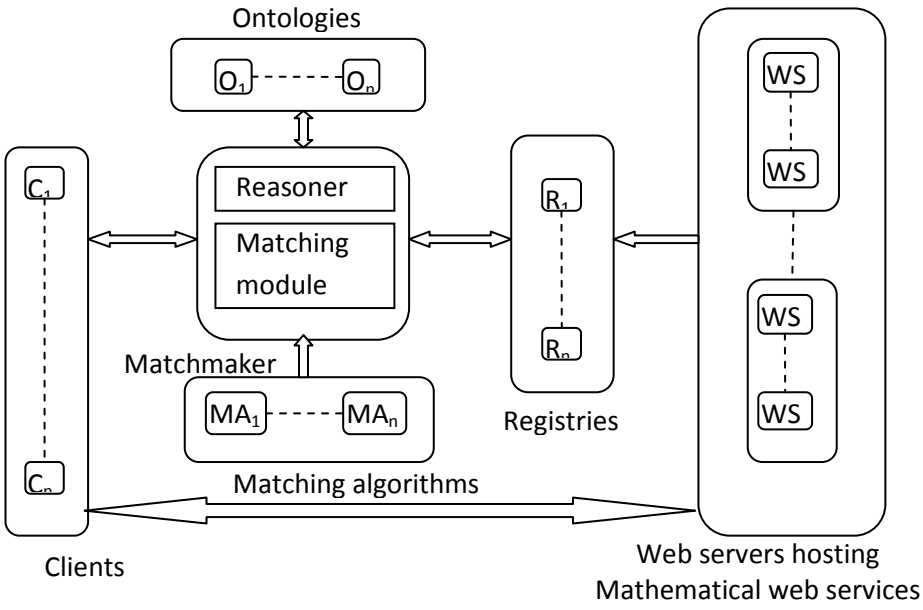


Figure III. 8. L'architecture du matchmaking

La recherche s'effectue par le matchmaker qui charge l'algorithme spécifié par l'utilisateur, ensuite le matchmaker contacte le raisonneur qui charge l'ontologie correspondante. Enfin, une recherche dans les enregistrements des services est effectuée et les détails des résultats sont retournés à l'utilisateur par le matchmaker.

Le matching extrait les préconditions et les postconditions de la requête, puis il construit une requête SQL pour trouver les structures de OpenMath des preconditions et postconditions des services qui sont semblables à celles de la requête.

Mavaddat et al. proposent dans [41] une technique de composition des services web basée sur la structure de graphe, ils considèrent le langage de description OWL-S qui spécifie les propriétés du comportement du service web qui est spécifié en termes d'entrées, de sorties et de l'exécution de flux de données. Ils utilisent le formalisme interface automates pour modéliser le comportement du service web. Pour modéliser formellement un service web, ils utilisent un modèle basé sur des états semblables aux diagrammes des automates d'états finis, ce modèle est appelé *interface automata* (IA) et il décrit dans la figure suivante :

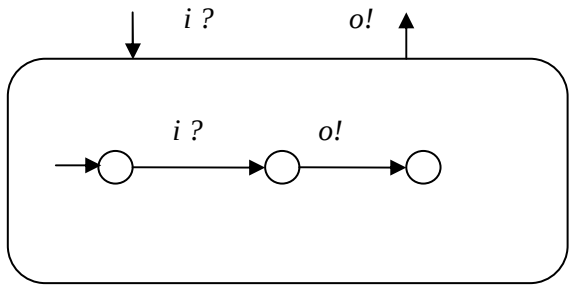


Figure III. 9. La représentation d'un service sous forme d'un automate d'états finis

Pour chaque service web trois types d'informations sont retenus :
WS= (A^{I}_{ws} , A^{O}_{ws} , k_{ws}), Avec : $A^{I}_{ws,}$ A^{I}_{ws} représentent respectivement l'ensemble des entrées/sorties d'un service web donnée et k_{ws} représente les dépendances entre ces derniers.
Les entrées et les sorties du service web ont respectivement les suffixes *?* et *!*
La figure précédente indique que le service web indiqué produit les sorties (*o*) si seulement si il reçoit en entrées (*i*) ; cette dépendance est directe entre ces entrées et sorties et elle est représentée par ($i \longrightarrow o$).

Les auteurs ont proposé un mécanisme pour résoudre le problème de la composition en stockant l'ensemble des informations de tout les services sous forme d'un graphe de dépendances définit comme suit : G= (V, E).
V étant les nœuds représentants l'ensembles des entrées sorties « A^{I}_{ws} , A^{O}_{ws}»
Un arc de $v_x \rightarrow v_y$ dans le graphe existe si et seulement s'il existe une dépendance entre les nœuds correspondants,
Une fois un modèle pour le service web est défini et aussi les services sont reliés entre eux sous forme d'un graphe de dépendances, un algorithme de recherche dans les graphes est utilisé pour trouver un chemin satisfaisant la requête.
Le problème revient alors à la recherche dans le graphe de dépendances l'ensemble des services web qui couvrent toute les dépendances données dans la requête après avoir déterminé le sous graphe des dépendances correspondant à la requête. Le service composite est construit d'une façon incrémentale avec les différents chemins retournés.

Medjahed [42] a présenté une technique basée sur la description déclarative de haut niveau pour générer le service composite. Cette technique utilise des règles de composition pour déterminer dans quelle mesure deux services sont composables. L'approche proposée se déroule en quatre phases :

- **Phase de spécifications** permet la description de haut niveau de la composition désirée en utilisant le langage CSSL (Composite Service Spécifications Language).
- **Phase de correspondance (matchmaking)** emploie les règles de composabilité pour générer des plans de composition conformes aux spécifications de la requête.
- **Phase de sélection** Si plus d'un plan est généré, alors le demandeur de service sélectionne un plan en se basant sur la qualité des paramètres de composition (QoC) (eg. coût, etc.).
- **Phase de génération** Une description détaillée du service composite est automatiquement générée et présentée au demandeur de service.

La principale contribution de cette approche est la notion de règles de composabilité. Les règles de composabilité prennent en considération les propriétés syntaxiques et sémantiques des services web.

Les règles syntaxiques incluent des règles pour les types d'opérations possibles et pour les liaisons protocolaires entre les services (i.e., les bindings).

Les règles sémantiques incluent le sous-ensemble suivant :

✓ La composabilité message définit que deux services web sont composable seulement si le message en sortie d'un service est compatible avec un message en entrée d'un autre service.

✓ La composabilité sémantique d'opération définit la compatibilité entre les domaines, catégories et objectifs de deux services web.

✓ La composabilité qualitatif définit les préférences du demandeur concernant la qualité des opérations pour le service composite.

Cette notion de règles de composabilité met en avant les attributs possibles des services web qui peuvent être utilisés dans la composition de services et peut ainsi jouer le rôle de directive pour d'autres méthodes de composition par planification.

Hamadi et Benatallah proposent dans [28] une algèbre basée sur les réseaux de pétri utilisée pour modéliser les flux de contrôle des services web. Cette algèbre est suffisamment expressive pour capturer la sémantique des services composés et leurs spécifications. Les réseaux de pétri sont utilisés pour modéliser et analyser plusieurs types de processus. Les auteurs considèrent le comportement des services web comme étant un ensemble d'opérations ordonnées qui peuvent être représentées par un réseau de pétri et représentent les transitions et les états du service représentent les nœuds. Ils décrivent la syntaxe et la sémantique informelle des opérateurs algébriques de service ainsi que les constructeurs qui permettent la combinaison des services et ils représentent l'ensemble des services sous forme d'une grammaire comme suite :

$$S ::= \varepsilon \mid X \mid S \odot S \mid S \oplus S \mid S \diamond S \mid \mu S \mid S \|_{\mathcal{C}} S \mid (S|S) \rightsquigarrow S \mid [S(p,q) : S(p,q)] \mid Ref(S,a,S)$$

Exemple de deux services composés séquentiellement :

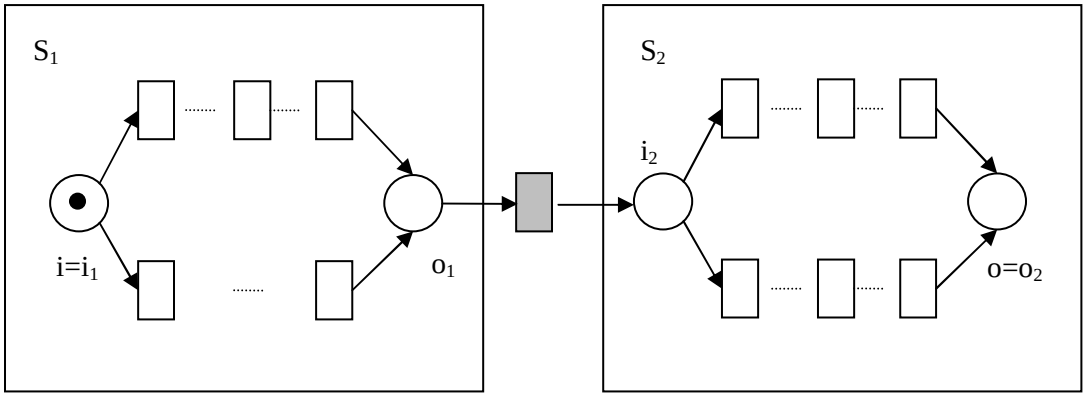


Figure III. 10. service $S_1 \odot S_2$

La grammaire précédente permet de définir différents types d'opérateurs de composition utilisés, nous trouvons l'opérateur de la composition séquentielle de deux services, la composition alternative, une séquence arbitraire, l'opérateur d'itération, l'opérateur du parallélisme …etc. graphiquement l'ensemble des services sont représentés par le réseau de pétri.

L'avantage de cette approche est que chaque service exprimé par les constructeurs de l'algèbre utilisés peut être transformé en représentation par le réseau de pétri et en utilisant les propriétés algébriques, on peut transformer et optimiser les expressions des services utilisées

au départ. En plus, utilisant un modèle formel nous permet la vérification des constructeurs et la détection des inconsistances à l'intérieur et entre les services.

Dans [8], Berardi et al. ont proposé une approche de composition basée sur le comportement du service web pour but d'automatiser le processus de composition de service, le comportement du service représente son exécution. Berardi représente toutes les exécutions possibles du service sous forme d'arborescence qui seront représentées sous forme d'un automate d'états finis non déterministe (FSM).

Les auteurs introduisent la notion de communauté de services qui est composée d'un ensemble de services qui partagent des actions, un service qui joint à la communauté doit exporter ses actions aux autres services en terme de l'alphabet de la communauté (actions communes), ces services se collaborent entre eux afin de construire un service composé. Un service appartenant à une communauté est vu de l'extérieur comme une boite noire qui expose un comportement représenté par une séquence d'actions atomiques de sa communauté et possédant des contraintes pour son invocation, la vision de ce service de l'intérieur est la façon que les actions appartenant à son comportement sont exécutées. Pour capturer ces deux vues du service, les auteurs définissent deux schémas : un schéma externe qui spécifie le comportement sous forme d'une arborescence d'actions et chaque instance de service exécute un chemin de l'arborescence, tandis que le schéma interne spécifie les informations que les instances de service exécutent chaque action et définit la notion de l'arborescence de l'exécution interne que les nœuds sont étiquetés par (*a, I*) où *a* est l'action à exécuter et *I* est l'ensemble de services qui exécutent *a*.

La figure suivante montre l'arbre d'exécution externe et l'arbre d'exécution interne :

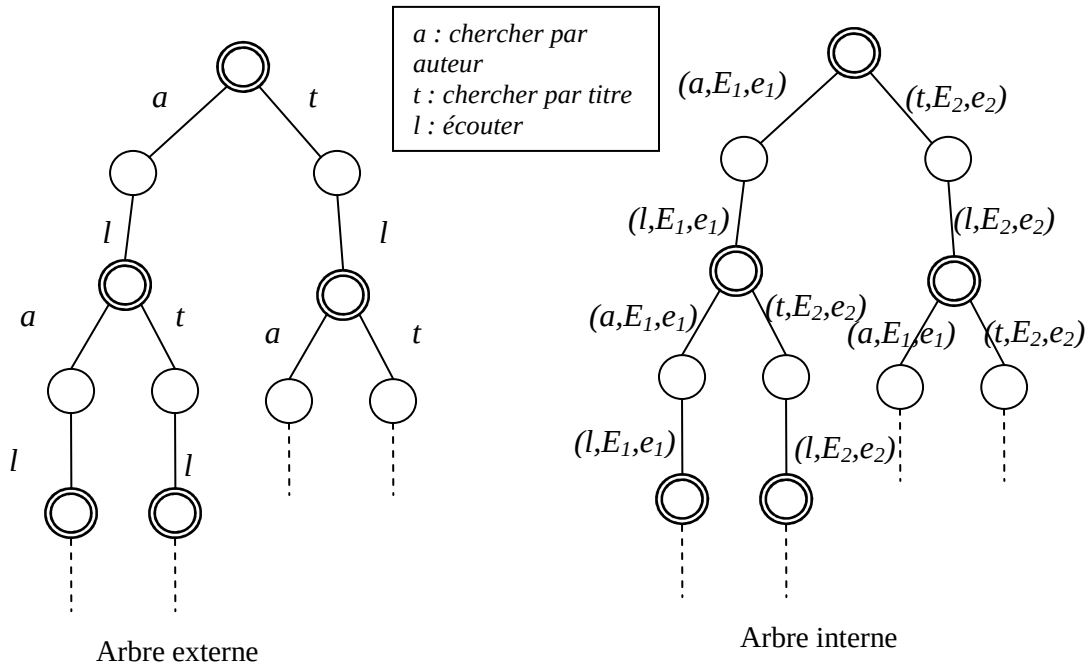


Figure III. 11. L'arborescence de l'exécution de service

Le problème de l'existence d'une composition revient à vérifier l'existence de l'arborescence de l'exécution interne qui est conforme au schéma de l'exécution externe et cohérent avec la communauté. Quand le client envoie sa requête, il l'exprime sous forme d'un schéma externe de service.

Un module composeur réalise une composition d'un service disponible pour l'exécution par la synthèse d'un schéma interne qui est une composition du service dans sa communauté. Les auteurs considèrent les schémas du service sont représentés par un automate d'états finis (FSM). Un schéma externe d'un service E est représenté comme suit :

A_E^{ext}= ($\sum$, S_E , s_E^0, δ_E, F_E) où : $\sum$ est l'alphabet de FSM, qui représente l'alphabet de la communauté, S_E est l'ensemble des états du FSM qui représente un ensemble fini d'états du service E, δ_E est la fonction de transition en exécutant l'action a, s_E^0 et F_E sont l'état initial et les états finaux respectivement.

La figure suivante illustre la modélisation de deux services sous forme de FSM.

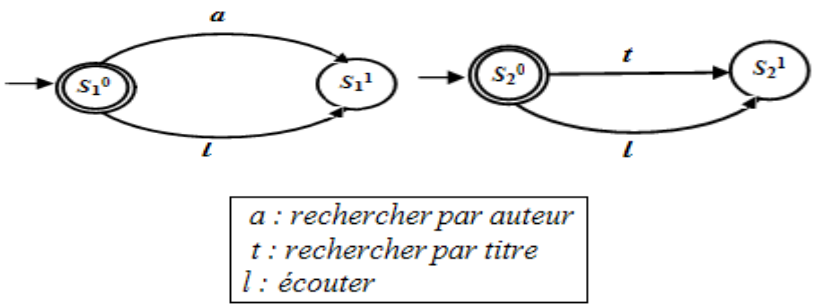


Figure III. 12. FSM représentant les services web recherche/auteur et recherche/titre

Le service désiré (requête) est spécifié de la même manière qu'un service web (FSM).

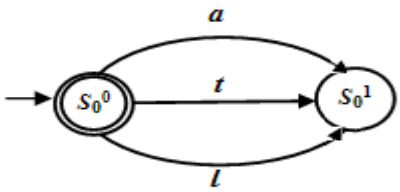


Figure III. 13. FSM du service désiré

Cette approche consiste à réduire le problème de la composition en un problème de satisfiabilité d'une suite de formule DPDL (Deterministic Propositional Dynamic Logic) [27] qui est une logique permettant d'exprimer des changements dans l'exactitude des formules logiques durant l'exécution d'un programme, elle est basée sur un ensemble de propositions (vraies ou fausses).

Après application de cette formule, Le résultat de la composition des deux services de la figure III.12, est illustré dans la figure suivante :

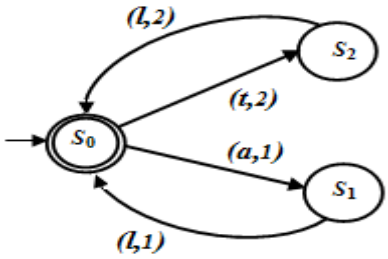


Figure III. 14. FSM du service web composite.

La principale contribution est de proposer un modèle formel pour la représentation des services électroniques et de venir par une approche de composition automatique basée sur les FSM, ce travail est un essai pour fournir un algorithme pour la synthèse de composition de services mais l'inconvénient est d'avoir une complexité très élevée et exponentielle.

III.5. Conclusion

Ce chapitre présente un état de l'art sur notre problématique, nous avons défini les concepts et les notions de bases pour le matching des services web, qui se situent sur la comparaison entre le matchmaking et les brokering approaches, une présentation générale sur les ontologies ainsi la description des approches utilisées dans la présentation des services. Dans une autre partie nous avons présenté les différents concepts et le modèle sur lesquels se base notre travail ; et nous finissons par la présentation de quelques travaux reliés au problème étudié. Dans le chapitre suivant, nous allons présenter les détails techniques et les notions utilisées dans notre algorithme.

CHAPITRE IV

L'algorithme du matching direct par substitution

Chapitre IV

L'algorithme du matching direct par substitution

IV.1. Introduction

Tari et Elgadawi [59,22] proposent un schéma du matching (FSMS, *The Functional Substitutability Matching Scheme*) qui utilise les fonctionnalités de services comme aspect de comparaison et la substitution comme principe de matching.

Dans ce chapitre, nous proposons un algorithme du *matching* des spécifications fonctionnelles par substitution de contraintes qui repose sur FSMS puis nous proposerons une approche dynamique de composition de services web, basée sur les capacités d'agent cognitifs. Nous considérons dans notre approche que le processus du matching des services est décentralisé car il s'effectue d'une façon indépendante et parallèle pour chaque service participant à la composition, ce qui nous donne la possibilité d'implémenter l'approche dans un environnement distribué, alors que le processus de la composition s'effectue d'une manière centralisée au niveau d'un seul agent appelé agent médiateur qui reçoit les résultats du matching de chaque service et cherche un bon plan de composition.

Avant de détailler notre algorithme, nous devons expliquer quelques notions et donner quelques notations utilisées ;

Les fonctionnalités d'un service web sont déterminées selon son comportement qui est extrait du modèle G+ [59]. De ce fait, le service est représenté comme une séquence d'opérations qui transforme les *pré-contraintes* aux *post-contraintes*. L'ensemble des contraintes qui apparaissent entre les opérations s'appellent *les contraintes descriptives*.

En général, quand l'opération Op_x termine son exécution, les contraintes à cet instant peuvent être classées en : Nouvelles, identiques et indépendantes contraintes. L'union de ces contraintes s'appelle les contraintes persistantes au point x notées f_x. Ces dernières doivent satisfaire les pré-contraintes de l'opération Op_{x+1} (Op_{x+1}^{Pre}), mais ça ne veut pas dire que toutes les contraintes de f_x sont nécessaires pour satisfaire Op_{x+1}^{Pre} car il peut y avoir des

contraintes indépendantes à Op_{x+1}^{Pre}. On classe f_x en contraintes effectives et contraintes libres (*effective constraints* f_x^e, *idle constraints* : f_x^i).

IV.2. L'algorithme proposé

L'algorithme de matching direct des services web fonctionne de la manière suivante :

Les informations relatives aux fonctionnalités sont extraites des requêtes des utilisateurs et les descriptions des services puis le *principe du matching* (la substitution) est appliqué sur ces informations. Le matching est considéré exact lorsque le but de l'utilisateur et celui du service sont atteints.

IV.2.1. Préliminaire

Le comportement interne du service est défini comme étant la séquence d'opérations ainsi l'ensemble des contraintes à chaque point de transition entre les opérations.

***Définition 8* : Le comportement et la fonctionnalité du service.**

Un état du comportement S_x en point de transition x entre les deux opérations Op_x et Op_{x+1} est défini comme le tuple $\langle f_x^e, f_x^i \rangle$. Le comportement du service web (β) est défini comme séquence $\langle S_0, S_1, \ldots\ldots, S_n \rangle$ ou l'ensemble des états élémentaires. D'où la fonctionnalité du service est définie comme$\langle GCtxtDesc, \beta \rangle$, GCtxtDesc étant l'ensemble des *contraintes descriptives* et β le comportement du service

IV.2.1.1. Le matching par le principe de la substitution

Etant donné les fonctionnalités demandées par l'utilisateur qui sont représentées par un ensemble de contraintes, le *matchmaker* doit trouver le service qui satisfait ces contraintes d'où le problème du matching est un problème de satisfaction de contraintes. Dans ce problème, on a besoin en générale de déterminer si un ensemble de contraintes source peut satisfaire un autre ensemble de contraintes de destination.

Dans les approches *satisfaction directe de contraintes (constraint direct satisfiability)*, les deux contraintes possèdent le même domaine (*constraint scope*). Alors que les approches *satisfaction indirecte de contraintes (constraint indirect satisfiability)* s'intéressent aux contraintes possédant des domaines différents.

La satisfaction indirecte de contraintes consiste à trouver une transformation $\Im$ qui transforme la contrainte de la source qui n'a pas le même domaine avec la contrainte destination à une autre contrainte intermédiaire de même domaine avec la contrainte destination en utilisant la sémantique. Cette transformation sert à la médiation entre les contraintes qui n'ont pas le même domaine [22].

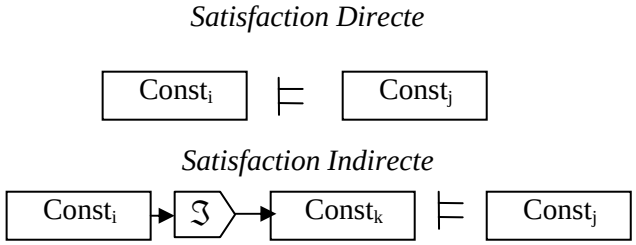


Figure IV. 1. Satisfaction Directe et satisfaction indirecte des contraintes

La figure suivante décrit le processus de réalisation de la satisfaction indirect des contraintes et les différentes transformations qui permettent de créer la contrainte intermédiaire.

Soient $Const_i$ et $Const_j$ deux contraintes. Si $Const_i$ satisfait indirectement $Const_j$, ceci est dénoté $Const_i \gg Const_j$.

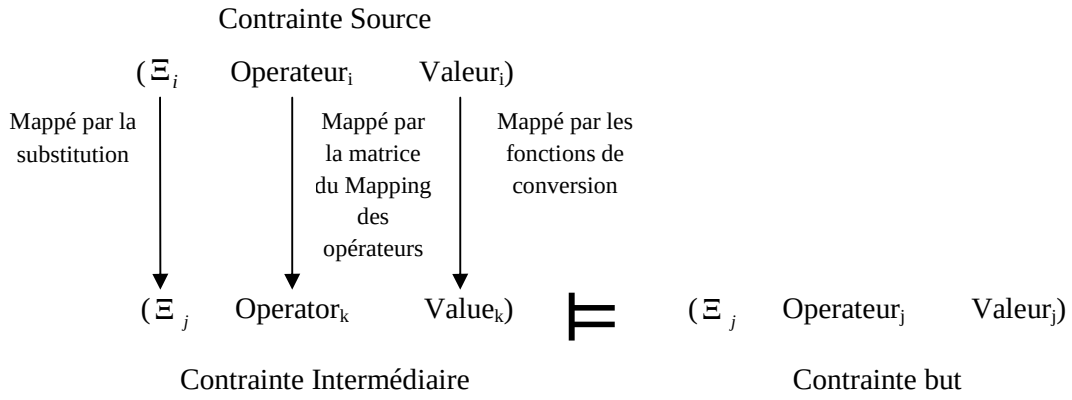


Figure IV. 2. La satisfaction indirecte des contraintes

La contrainte intermédiaire est créée de la manière suivante :

- Le domaine de la contrainte intermédiaire est le même domaine de la contrainte destination, c'est un domaine résultat de la substitution du domaine de la contrainte source et cela lorsqu'il existe un segment ou un chemin entre les deux domaines dans le graphe de substitution des concepts (CSG).
- Les valeurs de la contrainte intermédiaire sont les valeurs résultantes de l'invocation de la fonction de conversion correspondante avec les valeurs de la contrainte source.
- Les opérateurs de la contrainte intermédiaire sont les opérateurs correspondants aux opérateurs de la contrainte source dans la matrice de mapping des opérateurs (CMM).

IV.2.1.2. La substitution des contraintes

Définition 9 :

Soient $Const_i$, $Const_j$ et un but G. $Const_j$ peut être substituée par $Const_i$ (dénoté $(Const_i \trianglerighteq_G Conts_j)$) si $(Const_i \vDash Const_j) \vee (Const_i \gg_G Const_j)$.

La substitution d'un ensemble de contraintes

Soient deux ensembles de contraintes f_i, f_j et un but G. f_j peut être substituée par f_i en respectant G (dénoté $(f_i \trianglerighteq_G f_j)$) si $\forall Const_u \in f_j, \exists Const_v \in f_i$ tel que $Const_v \trianglerighteq_G Conts_u$.

Un but est considéré atteint lorsque les pré-contraintes de HLFC sont transformées aux post-contraintes de HLFC via le modèle du comportement et que les contraintes descriptives sont satisfaites.

***Définition 10* : La réalisation d'un but**

Soit un but *G* décrit par un GAP ($Gp = < Op, Ctxt, OpSeq >$). G est réalisé (défini par $G \vdash < S_i, T_i, \beta_i, D_i >$) lorsque T_i et D_i sont satisfaites tel que $\sigma_i(S_i) = T_i$, $S_i = GCtxt^{Pre}$, $T_i = GCtxt^{Post}$, $D_i = GCtxt^{Desc}$, σ_i est la fonction de transformation correspondante à β_i. $\vdash$ est l'opérateur d'accomplissement de but.

IV.2.2. Les détails le la proposition

Notre algorithme repose sur le *matching* d'états de deux comportements, à savoir celui du service et celui de la requête de l'utilisateur. Le matching des deux séquences d'états peut être déterminé en utilisant deux approches à savoir : L'approche *one-to-one* et l'approche *m-to-n*.

La première approche exige que les séquences d'états aient la même longueur et elle utilise l'analyse syntaxique. Les approches *m-to-n* sont utilisées afin de surmonter les lacunes des premières, en réalisant le matching basé sur la sémantique et la séquence d'état.

Nous proposons un algorithme qui accepte deux séquences d'états dans les deux cas et peut être ainsi utilisé pour les deux approches citées.

L'algorithme proposé s'opère en trois étapes :

1- Construction des tables du matching pour chaque état dans la séquence de destination (de but): L'algorithme essaye de trouver pour chaque état but tous les *matching* possibles avec tous les états sources, et avec toutes les combinaisons de clusters formés à partir des états sources (la définition de cluster d'état et la notion d'extension (fusion) d'états seront expliquées plus loin). La valeur de la matrice du matching est égale à 1 si un matching existe entre l'état *objectif* et l'état *source* ou le cluster correspondant et 0 dans le cas contraire.

2- Le parcours des tables de matching et la détermination de la liste de cluster pour chaque séquence d'état de chaque comportement. Ensuite, nous vérifions les deux séquences d'états. S'il existe des états *sources* non matchés, alors on ajoute chaque état de ces derniers au cluster précédemment matché (un cluster peut contenir au minimum un état), c'est-à-dire que si nous avons un état source S_j non matché, l'algorithme cherche un état matché S_i tel que i<j et il n'existe pas un état S_k matché tel que i<k<j, ensuite l'algorithme fusionne l'état S_j avec le cluster qui contient l'état S_i. Deuxièmement, si tous les états *but* sont matchés alors *matching est terminé*, sinon nous procédons à la troisième étape.

3- Fusionner chaque état objectif avec ses prédécesseurs et essayer de trouver un cluster qui sera matché avec un autre cluster de la séquence d'états source.

***Définition 11*: Le matching d'état**

Soient l'état de la source $S_x = <f_x^e, f_x^i>$, l'état de l'objectif $S_t = <f_t^e, f_t^i>$ et deux comportements β_i et β_j. Un objectif G, sachant que S_x appartient au comportement β_i et S_t appartient au comportement β_j. L'état source S_x peut réaliser le matching vers S_t en respectant G (dénoté $S_x \trianglerighteq_G S_t$) si ($f_x^e \trianglerighteq_G f_t^e$).

Nous définissons le domaine d'un état $S_x = <f_x^e, f_x^i>$ comme étant le domaine de ses contraintes effectives ($\Xi(f_x^e)$) dénoté par (ΞS_x).

***Définition 12*: L'extension d'état.**

Dans l'approche *m-to-n*, lorsqu'un état source S_x ne peut être matché avec un état objectif S_t, on examine alors la possibilité d'un matching résultant de la fusion de S_x et S_t avec d'autres états dans leur séquence correspondante. Ce cas est dit extension d'états (*state expansion*).

Soit un état $S_i \in \beta_x$, S_i est étendu à S_i' lorsqu'il est fusionné avec d'autres états adjacents dans β_x formant un cluster d'états. Les opérations des états fusionnés sont elles mêmes fusionnées formant ainsi l'opération correspondante à l'état S_i'.

Nous illustrons dans les deux figures suivantes la fusion des états et leurs opérations correspondantes.

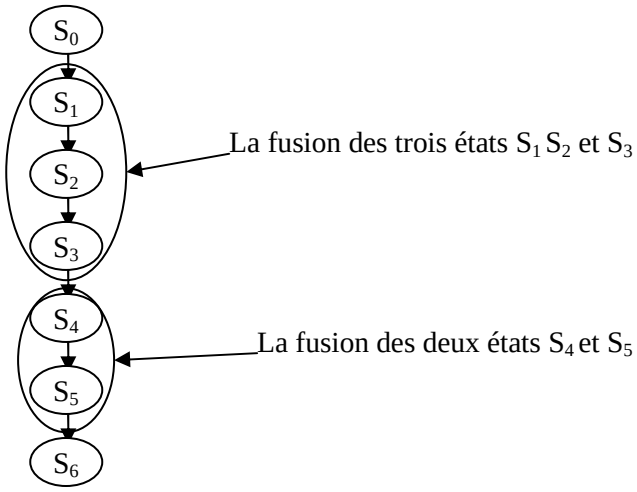


Figure IV. 3. La fusion d'états

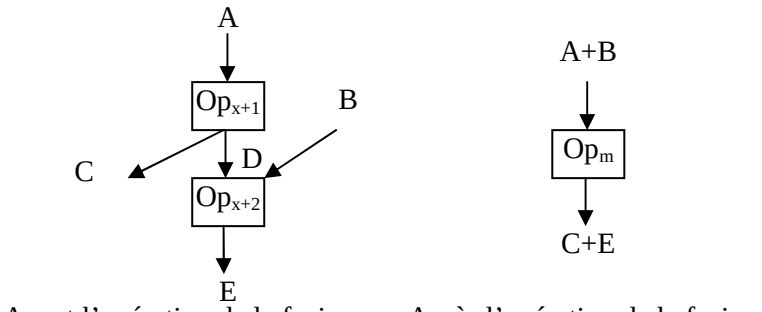


Figure IV. 4. La fusion des opérations

La figure IV.3 montre une séquence constituée de sept états, après la fusion des trois états S_1, S_2, et S_3 dans un seul cluster et les deux états S_4 et S_5, nous aurons une séquence de quatre états. La figure IV.4 décrit la fusion de deux opérations consécutives Op_{x+1} et Op_{x+2} formant l'opération Op_m qui est formée en cascadant Op_{x+1} et Op_{x+2}, cette figure indique que les concepts d'entrées de Op_m est l'union entre l'ensemble de concept A et B, ses concepts de sorties est l'union entre les concepts C et E, alors que l'ensemble de concepts D n'apparaissent pas ni dans les concepts d'entrée ni dans les concepts de sortie de Op_m. Cet ensemble de contraintes D sont des contraintes résultantes de l'exécution de l'opération Op_{x+1} et elles sont en même temps des précontraintes de l'opération suivante. Ces contraintes disparaissent lors de la fusion des deux opérations. Cette figure montre aussi un autre type de contraintes qui sont les contraintes libres (*idle constraints*) représentées par l'ensemble C. Cet ensemble est le résultat de l'exécution de Op_{x+1} mais elles ne participent pas à l'invocation de l'opération suivante. Nous illustrons plus clairement la fusion de deux états avec leurs opérations correspondantes dans la figure suivante.

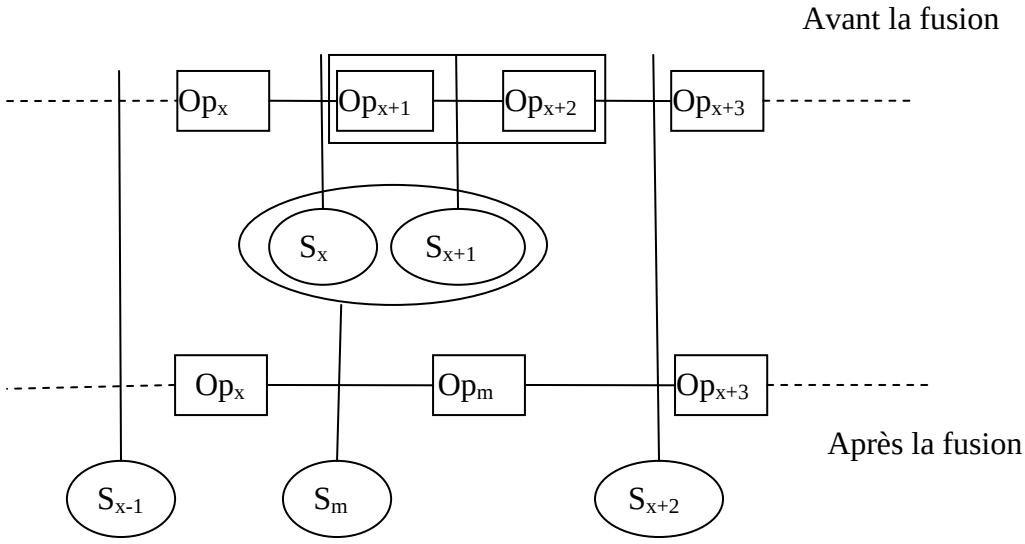


Figure IV. 5. La fusion de deux états avec leurs opérations correspondantes

Lorsque deux états $S_x = \langle f_x^e, f_x^i \rangle$ et $S_{x+1} = \langle f_{x+1}^e, f_{x+1}^i \rangle$ sont fusionnés ainsi que leurs opérations correspondantes Op_{x+1} et Op_{x+2}, nous aurons l'état résultat S_m et l'opération résultantes Op_m définis comme suit :

$$Op_m^{Pre} = Op_{x+1}^{Pre} + (Op_{x+2}^{Pre} \lozenge Op_{x+1}^{Post}) \wedge \; Op_m^{Post} = Op_{x+2}^{Post} + (Op_{x+1}^{Post} \lozenge Op_{x+2}^{Pre}) \text{ et} \qquad \text{(I)}$$

$$f_m^i = \left(f_x^i \lozenge f_{x+1}^e\right) \text{et} \;\; f_m^e = \left(f_x - f_m^i\right). \qquad \text{(II)}$$

Notre algorithme est basé sur les théorèmes suivants pour former les clusters.

Théorème 1:

Soit un état source (ou un état résultat de la fusion d'un ensemble d'états sources) $S_x = \langle f_x^e, f_x^i \rangle$ et un état but $S_t = \langle f_t^e, f_t^i \rangle$, si S_x matche S_t alors si $\exists$ un successeur de S_x et si les deux ensembles de contraintes $f_x^e et f_{x+1}^e$ sont indépendants, alors $S' = S_x + S_{x+1}$ matche S_t.

Preuve :

$S_x \trianglerighteq S_t$ si et seulement si $f_x^e \trianglerighteq f_t^e$ (*définition 11*). D'après (II), nous avons :

$$S' = S_x + S_{x+1}$$

$$f'^e = f_x - f'^i \;\text{ et } f'^i = f_x^i \lozenge f_{x+1}^e.$$

Nous avons $f_x = f_x^e + f_x^i$ => $f'^e = f_x^e + f_x^i - (f_x^i \lozenge f_{x+1}^e)$

Posons, $\alpha = f_x^i - (f_x^i \lozenge f_{x+1}^e)$ alors $f'^e = f_x^e + \alpha$

D'après la *définition 10*, $f_x^e \trianglerighteq f_t^e$ => $\forall const_t \in f_t^e, \exists const_x \in f_x^e$, tel que $const_x \trianglerighteq const_t$. Nous avons $\forall$Z un ensemble de contraintes, alors $f_x \subseteq f_x + Z$. Ce qui veut dire que $\forall const_x \in f_x^e \Longrightarrow const_x \in f_x^e + Z$,

Nous déduisons que si $f_x^e \trianglerighteq f_t^e \Longrightarrow (f_x^e + Z) \trianglerighteq f_t^e \quad \forall Z$ ensemble de contraintes indépendantes à f_x^e , en remplaçant Z par α nous donne $f'^e \trianglerighteq f_t^e$, alors l'état du matching du cluster pour le même état but ne change pas lorsque $f_x^e \; et \; f_{x+1}^e$ sont indépendants.

Théorème 2:

Soient un état source (ou un état résultant de la fusion d'un ensemble d'états sources) $S_x = \langle f_x^e, f_x^i \rangle$ et deux états but consécutifs $S_t = \langle f_t^e, f_t^i \rangle$ et $S_{t+1} = \langle f_{t+1}^e, f_{t+1}^i \rangle$. Si S_x matche avec $S_t$et S_x matche avec S_{t+1} , alors S_x matche avec l'état résultat de la fusion de S_t et S_{t+1}.

Preuve :

Nous avons :

$S_x \trianglerighteq S_t$ alors $f_x^e \trianglerighteq f_t^e$ ……..(I).

$S_x \trianglerighteq S_{t+1}$ alors $f_x^e \trianglerighteq f_{t+1}$……(II).

$S' = S_t + S_{t+1} \Longrightarrow f'^e = f_t^e + (f_t^i - (f_t^i \lozenge f_{t+1}^e))$.

Démontrons que $f_x^e \trianglerighteq f'^e$

Nous avons selon la définition de la différence sémantique *(définition 7)* les deux cas suivants :

1) Si toutes les contraintes $Op_{x+1}^{pre} \subseteq Op_x^{post}$ alors

$f_t^i - (f_t^i \lozenge f_{t+1}^e) = \emptyset \Longrightarrow f'^e = f_t^e$…… (III).

I et III $\Longrightarrow f_x^e \trianglerighteq f'^e$ d'où S_x matche le cluster formé des deux états S_t et S_{t+1}.

2) S'il existe de nouvelles contraintes, nous aurons

$(f_t^i - (f_t^i \lozenge f_{t+1}^e)) \vDash f_{t+1}^e$……...(IV).

II et IV$\Longrightarrow f_x^e \trianglerighteq (f_t^i - (f_t^i \lozenge f_{t+1}^e))$……..(V).

I et IV $\Longrightarrow f_x^e \trianglerighteq f_t^e \cup (f_t^i - (f_t^i \lozenge f_{t+1}^e)$ d'où $f_x^e \trianglerighteq f'^e$

Ce qui nous démontre que S_x matche le cluster formé de (S_t et S_{t+1}).

Théorème 3:

Soient deux états sources consécutifs (ou deux états consécutifs résultants de la fusion de deux ensembles d'états sources) $S_x = \langle f_x^e, f_x^i \rangle$ et $S_{x+1} = \langle f_{x+1}^e, f_{x+1}^i \rangle$ et un état but $S_t = \langle f_t^e, f_t^i \rangle$

Si S_x matche avec $S_t$et S_{x+1} matche avec S_t , alors l'état résultat de la fusion de $S_x \; et \; S_{x+1}$ matche avec S_t.

Preuve :

$S_x \trianglerighteq S_t \Longrightarrow f_x^e \trianglerighteq f_t^e$……. (I).

$S_{x+1} \trianglerighteq S_t \Longrightarrow f_{x+1}^e \trianglerighteq f_t^e$…….. (II).

$S' = S_x + S_{x+1} \Longrightarrow f'^e = f_x^e + (f_x^i - (f_x^i \lozenge f_{x+1}^e))$.

1) $f_x^i - (f_x^i \lozenge f_{x+1}^e) = \emptyset$ alors $f'^e = f_x^e$

$\Longrightarrow f'^e \trianglerighteq f_t^e$ d'où le cluster matche S_t

2) f_x^i - (f_x^i ◊ f_{x+1}^e) ≠ ∅ ⟹ (f_x^i - (f_x^i ◊ f_{x+1}^e))⊨ f_{x+1}^e……..(III).

De II et III ⟹ f_x^i - (f_x^i ◊ f_{x+1}^e) ⊨ f_t^e ……..…(IV).

I et IV ⟹ (f_x^e ∪ (f_t^i- (f_t^i ◊ f_{t+1}^e)) ⊵ f_t^e d'où

f'^e ⊵ f_t^e d'où le cluster (S_x , S_{x+1}) matche S_t .

L'algorithme

Cette première partie de l'algorithme sert à construire la table de matching nommée *match_table*. Cette table sert à sauvegarder l'état du matching pour tous les clusters qu'on peut former à partir de la séquence d'états de la source avec chaque état de la séquence d'états de l'objectif. *match_table* est une matrice carrée de n lignes, où n est le nombre d'états dans la séquence d'états de la source. Pour chaque état but T_j, l'algorithme calcul sa matrice correspondante.

Match_table [i , k] signifie l'état du matching du cluster formé de l'état source S_i jusqu'à l'état source S_k dans la séquence d'états avec l'état de l'objectif correspondant, une table est construite pour chaque état, alors si nous avons m états de l'objectif, l'algorithme calcule m table de matching. Ensuite, un calcul sera fait pour déterminer la paire du matching constituée du cluster d'états source avec l'états objectif correspondant à cette matrice (Algorithme 2). On sauvegarde ensuite la paire matchée (formée du cluster d'état source et l'état but) dans la table de cluster (*table_cluster*).

La table de cluster est une liste chainée, chaque élément de cette liste contient deux champs : *begin_cluster* qui signifie la référence de l'état source qui représente le début du cluster et *end_cluster* qui signifie l'état de la fin de cluster. Chaque élément de la liste représente le cluster qui matche un état objectif correspondant à cet élément, ce qui veut dire que la taille de la liste chainée est égale au nombre d'états objectif.

L'ensemble *unmatched* contient tous les états objectifs qui ne sont pas matchés. Algorithme 3 effectue la fusion de chacun de ces états avec leurs prédécesseurs qui sont déjà matchés, et essaye de trouver un matching pour le cluster créé.

Algorithme 1 : le matching basé sur le comportement

Entrées : « n » : le nombre d'états source, « m » : le nombre d'états but
Un tableau d'états source (S) ; Un tableau d'états but (T).
β_i : Le modèle de comportement de service.
β_j : Le modèle de comportement de l'utilisateur.

Sorties : Deux séquences d'états, un ensemble d'états but non matché

Début

Variable *Cluster {borne_inf, borne sup}, S: état, Unmatched= ∅*
Table_Cluster { Begin_Cluster, End_Cluster} : Tableau [n]

Cluster [borne_inf, borne_sup] = [0,0]
Pour chaque état_but T_j **faire**
Initialiser Match_Table = 0
Pour chaque etat_source S_i **faire**
Si ($S_i \trianglerighteq T_j$) **alors**
Match_Table [i , i] = 1
Fin si
Si (il existe un successeur pour S_i , i <n-1) **alors**
S = S_i K= i+1
Pour chaque état S_K **faire**
S=S+S_K // fusionner S avec S_K
Si ($S \trianglerighteq T_j$) **alors**
Match_Table [i, k] = 1
Fin si
k = k+1
Fin pour
Fin si
Fin pour
Cluster [borne_inf, borne_sup]=Algorithme 2 (Match_Table, cluster [borne_inf, borne_sup],j)
// sauvegarder le cluster pour l'état_but j
Si ((cluster. borne_inf ≠ -1) et (cluster. borne_sup) ≠ -1) **alors**
Table_cluster [j] . Begin_cluster = cluster . borne_inf
Table_cluster [j] . End_cluster = cluster . borne sup
Fin si
Fin pour
// parcourir table_cluster
Liste_cluster_service → cluster → Begin = T [0]. Begin_cluster
Liste_cluster_service → cluster → end = T [0]. End_cluster
Pour i allant de 1 à (m-1) **faire**
Si T[i]. Begin_cluster ≤ T[i-1]. end _cluster **alors**
T[i].Begin_cluster = MIN (T[i-1].Begin_cluster, T[i].Begin_cluster)
T[i-1].Begin_cluster = T[i].Begin_cluster
T[i].End_cluster = MAX (T[i-1]. End_cluster, T[i]. End_cluster)
T[i-1]. End_cluster = T[i]. End_cluster
Sinon *// clustrer tout les états sources non matchés avec leurs prédécesseurs*
S_k = T[i].Begin_Cluster
T[i-1].end_cluster=S_{k-1}
Fin si
Fin pour

Pour i allant de 0 à m-1 **faire** // générer cluster_service
Si $T_i \notin$ Unmatched **alors**
Liste_cluster_service → cluster → Begin = T[i].Begin_cluster
Liste_cluster_service → cluster → end = T[i].end_cluster
Liste_cluster_service → cluster = Liste_cluster_service → suivant
Fin si
Fin pour
Pour i allant de 0 à m-1 **alors** // Clustrer les états but,
Si $T_i \notin$ Unmatched **alors**
Liste_cluster_but →cluster → Begin = T_i
i = i+1
Tant que (T[i].Begin_cluster = T[i-1].Begin_cluster) et (i≠m)**alors**
i = i+1
Fin tant que
Liste_cluster_but → cluster → end = T_i
Liste_cluster_but → cluster = liste_cluster_but → suivant
i = i+1
Fin si
Fin pour
Pour tout $T_i \in$ Unmatched **faire**
Si i < m **alors**
Sortir =vrai, j= i+1
Tant que ((j ≤ m-1) et sortir) **faire**
Si $T_j \in$ unmatched **alors**
Si j= (m-1) aller à *Etiquette*
j = j+1
Sinon
Algorithme 3 (T[j].end_cluster, T_i)
Sortir = faux
Fin si
Fin tant que
Sinon
Etiquette : sortir= vrai, j= i-1
Tant que j > 0 et sortir **faire**
Si $T_j \in$ Unmatched **alors**
j = j-1
Sinon
Algorithme 3(T[j].end_cluster, T_i)
Fin si
Fin tant que
Fin si
Fin pour
FIN

Algorithme 2 : le calcul du cluster *service* correspondant à chaque état *but*
Entrée : Match_Table , cluster [borne_inf, borne_sup], T, Unmatched {}
Sortie : Cluster [borne_inf, borne_sup]
Début :
Pour i allant de (cluster.borne_inf) à (cluster.borne_sup) **faire**
Si Match_Table [i, cluster.borne_sup] = 1 **alors**
retourne (Cluster(i, cluster.borne_sup))
Fin si
Fin pour
Si (cluster.borne_sup < n-1) **alors**
Pour j allant de (cluster.borne_sup+1) à (n-1) **faire**
Pour i allant de j à 0 **faire**
Si Match_Table [i,j] = 1 **alors**
retourne Cluster (i,j)
Sinon
i = i-1
Fin si
Fin pour
Fin pour
Fin si
Marquer T non matché
Unmatched = unmatched+ T
retourne cluster (-1, -1)
FIN

Algorithme 3 : le matching des états but en les fusionnant inversement
Entrée : S_i , T_j
Sortie : Deux listes de cluster d'états, $Table\_Cluster_i$ (le cluster qui matche chaque états)
Unmatched *//ensemble d'états unmatchés.*
// Essayer de trouver pour chaque état but non matché un matching en le fusionnant avec des prédécesseurs
Debut :
matché = faux , S = S_i , T= T_j
Pour k allant de j-1 à 0 et ($\overline{matché}$) **faire**
T=T+T_k
Cluster_etat_but = cluster (T_k à T_j)
Fusionner les cluster_source qui matchent les états but de T_k à T_j
S = Résultat de la fusion de tous les états source du nouveau cluster
Si S ⊵ T **alors**
Mettre à jour les deux listes de cluster source et but
Mettre à jour la table cluster de chaque état T_i ($Table\_Cluster_i$)
matché = Vrai
Unmatched = Uunmatched – (état_but_cluster)
Fin si
Fin pour
FIN

IV.2.3. La complexité de l'algorithme

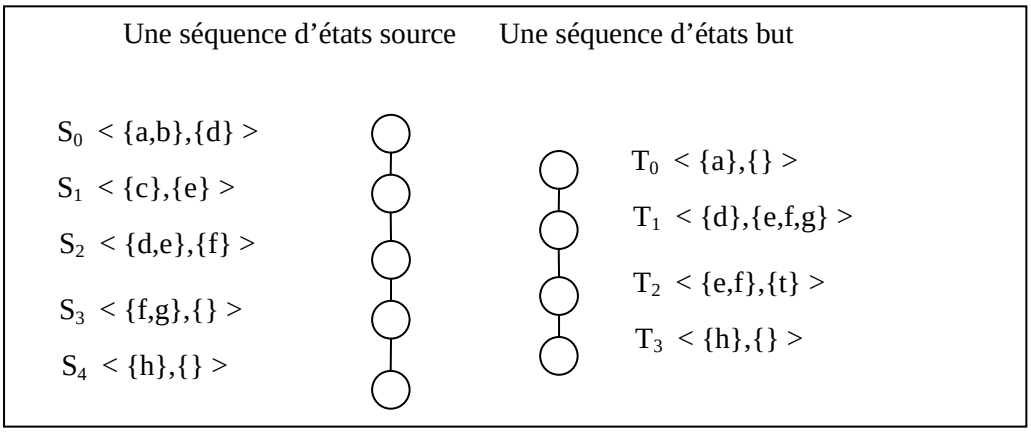
La complexité de l'algorithme est de $O(\frac{n^3}{2})$. En effet, la complexité de notre algorithme réside à la détermination de l'état du matching pour un état but T, qui est le calcul de la table de matching (*match_table*). Cela se fait en fusionnant chaque état source avec son successeur un par un pour former les clusters. L'algorithme commence à vérifier le matching du premier état source avec l'état but correspondant, puis il procède à la formation les clusters avec ses successeurs un par un et vérifier en même temps leur état du matching. Si on considère le nombre d'états source est n, alors pour former tous les clusters contenant le premier état nécessite n opération. Ensuite l'algorithme procède au deuxième état de la même façon que le premier, ainsi de suite jusqu'à terminer toute la séquence d'états source. Alors le nombre d'opération est de n+(n-1)+…+1 qui est égale à $\frac{n(n+1)}{2}$ ……(1). Si l'état de l'objectif n'est pas matché, il sera fusionné avec ses prédécesseurs, ce qui nous donne une complexité de ce deuxième cas est simplement $O(n)$ ……(2), alors à partir de (1) et (2), la complexité finale pour chercher un matching pour un état est de $O(\frac{n^2}{2})$, en répétant cela pour chaque état dans la séquence but, cela nous donne la complexité totale qui est égale à $O(\frac{n^3}{2})$.

Cet algorithme est une amélioration de l'algorithme proposé par Tari et Elgedawy. Notre algorithme est de complexité inferieure à ce dernier. L'algorithme proposé par Tari est de complexité égale à $O(n^4)$ alors que dans ce travail, nous avons réduit cette complexité en calculant l'état du matching de tous les clusters. Cela nous permet d'avoir l'état du matching d'un état source fusionné avec ses prédécesseurs avec l'état de but dans le cas où l'algorithme ne trouve pas un matching de l'état source concerné fusionné avec ses successeurs. Ce qui nous permet d'éviter l'étape *reverse expansion* appliquée dans l'algorithme proposé par Tari et al. *Reverse expansion* est la fusion de l'état avec ses prédécesseurs dans la séquence d'états, alors que *down expansion* est la fusion d'un état avec ses successeurs.

Si nous considérons que les contraintes effectives des états source sont indépendantes, la complexité de la construction de la table de matching sera réduite considérablement et cela en appliquant le théorème 1. Il suffit pour cela de trouver un matching entre un état source et un état but. Nous considérons directement que tous les clusters qui seront formés de cet état source vont réaliser le matching avec cet état but.

IV.3. Un exemple d'application

Considérant les deux séquences d'états source et but suivantes :



L'algorithme procède par le calcul des tables de matching pour chaque état but. Le résultat du calcul est donné dans les tableaux ci-dessous. Dans notre exemple, nous avons représenté les contraintes avec des lettres, alors deux contraintes sont substituables si elles sont représentées par la même lettre, exemple : l'état S_x<{a,b},{c,d}> matches S_t<{a,b},{e,f}>

Les tables du matching sont les suivantes :

T_0	S_0	S_1	S_2	S_3	S_4
S_0	1	1	1	1	1
S_1		0	0	0	0
S_2			0	0	0
S_3				0	0
S_4					0

T_1	S_0	S_1	S_2	S_3	S_4
S_0	0	0	1	1	1
S_1		0	0	0	0
S_2			1	1	1
S_3				0	0
S_4					0

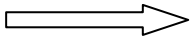
T_2	S_0	S_1	S_2	S_3	S_4
S_0	0	0	0	0	0
S_1		0	0	0	0
S_2			0	1	1
S_3				0	0
S_4					0

T_3	S_0	S_1	S_2	S_3	S_4
S_0	0	0	0	0	0
S_1		0	0	0	0
S_2			0	0	0
S_3				0	0
S_4					1

Après le parcours de ces tables de matching, l'algorithme nous donne la table_cluster suivante

	Begin_cluster	End_cluster
T_0	S_0	S_0
T_1	S_2	S_2
T_2	S_2	S_3
T_3	S_4	S_4

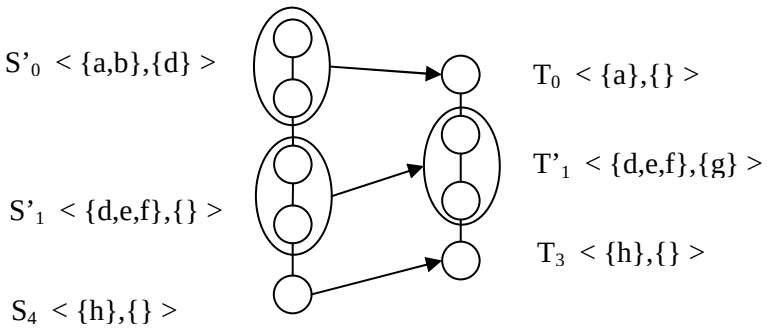
Ce tableau sera transformé en



	Begin_cluster	End_cluster
T_0	S_0	S_0
T_1	S_2	S_3
T_2	S_2	S_3
T_3	S_4	S_4

A l'étape suivante et après la création des clusters des états sources, l'algorithme trouve que l'état S_1 ne matche aucun état but, alors il sera clustré avec l'état précédent S_0 d'après le théorème 1. Nous remarquons que l'état T_1 et T_2 sont matchés par un seul cluster (S_2, S_3) ; d'après le théorème 2, T_1 et T_2 forment un seul cluster.

Après la terminaison de l'algorithme, nous obtiendrons deux séquences d'états de même taille.



Résultat de l'application de l'algorithme

IV.4. Conclusion

Ce chapitre est consacré au développement de l'algorithme que nous avons proposé. Il est basé sur les spécifications fonctionnelles de haut niveau. Il constitue une extension des travaux de Tari et Elgedawy [59,22] et les principes qu'ils ont utilisé dans l'approche proposée. Nous avons utilisé le principe de substitution de contraintes dans la vérification de la correspondance entre les spécifications de la requête et celles du service web.

L'algorithme proposé est d'une complexité inférieure à celui proposé par Tari et Elgedawy. De plus, il accepte des séquences d'états différentes ou égales. Il peut être utilisé pour les approches n-to-n ainsi les approches m-to-n. Ces dernières sont utilisées dans le cas où nous avons deux séquences de tailles différentes. L'algorithme recherche deux séquences de tailles égales en fusionnant des états dans les deux séquences. Les approches n-to-n essayent de trouver un matching direct entre les états, mais dans le cas où il y a des états qui ne peuvent être matchés, ces algorithmes renvoient un matching incomplet. Par contre, l'avantage de notre algorithme est qu'il cherche un matching pour ces états en les fusionnant avec d'autres états déjà matchés afin d'avoir un matching total.

Notre algorithme utilise des structures de données utilisées pour sauvegarder l'état du matching de chaque état objectif et de récupérer tous les états but qui ne sont pas matchés. Il nous permet de récupérer des segments (*chunk*) dans la liste des états but qui sont matchés par la liste des états sources dans le cas où la séquence des états source ne peut pas matcher toute la séquence des états but et de définir tous les états but qui ne peuvent pas être matchés. Ces structures nous permettent alors d'appliquer l'algorithme sur plusieurs séquences d'états source avec une même séquence d'états but indépendamment les unes des autres. Cette caractéristique nous donne une possibilité d'appliquer l'algorithme sur des machines distribuées sur des séquences d'états source différentes. A partir de ces avantages, nous avons l'idée de proposer une approche de composition dynamique qui applique l'algorithme que nous avons proposé au niveau des agents cognitifs qui représentent des séquences d'états des services web. Le chapitre suivant est consacré pour détailler notre approche de composition.

CHAPITRE V

Une approche dynamique pour la composition des services web

Chapitre V

Une approche dynamique pour la composition des services web

V.1. Introduction

L'un des problèmes les plus importants dans le processus de composition dynamique de services web est le problème du *matching*.

Le *matching* permet ainsi de sélectionner les services les plus appropriés pour répondre au besoin de l'utilisateur et d'intégrer ceux qui sont les plus aptes à communiquer pour réaliser une fonctionnalité donnée.

Dans cette deuxième partie, nous proposons une approche de composition des services web sémantique. Elle est basée sur les capacités d'agents cognitifs qui permettent l'utilisation de la représentation sémantique de service web ainsi que celle de la requête afin de réaliser un matching. Dans notre situation, les représentations sémantiques sont les contexts du scénario utilisé pour réaliser l'objectif du service et de l'utilisateur, à savoir les **pré-contraintes (**$Ctxt^{Pre}$**), describing-contraintes (**$Ctxt^{Desc}$**),** et les **post-contraintes (**$Ctxt^{Post}$**).**

Dans un environnement ouvert et dynamique comme internet, il est plus intéressant de réaliser une composition dynamique et à la demande que d'utiliser une composition statique.

L'inconvénient des approches d'agrégation séquentielle est qu'elles ont une complexité très importante et exponentielle. Cette complexité est égale au nombre d'états dans la séquence d'états de la requête puissance le nombre de service (n^m) ; n étant le nombre d'état de la requête et m le nombre de services. Le but de notre approche est de réduire cette complexité en la rendant linéaire.

V.2. Une approche de composition basée sur le système muti-agents

Notre approche est constituée de trois types d'agent :

– Un agent utilisateur qui émet sa requête à l'agent médiateur.

– Un agent médiateur qui reçoit la requête du client, extrait les différents contextes, et les diffuse aux agents fournisseurs.

– Un ou plusieurs agents fournisseurs (service) qui réalisent le matching et retransmettent le résultat au médiateur, qui va lui-même, réaliser et extraire un bon plan de composition et le renvoyer à l'utilisateur.

Nous donnons par la suite la définition de l'agent :

« Un agent est une entité autonome, réelle ou abstraite, qui est capable d'agir sur elle-même et sur son environnement, qui, dans un univers multi-agents, peut communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de ses connaissances et des interactions avec les autres agents »[23].

La figure suivante illustre l'architecture montrant la communication entre les agents :

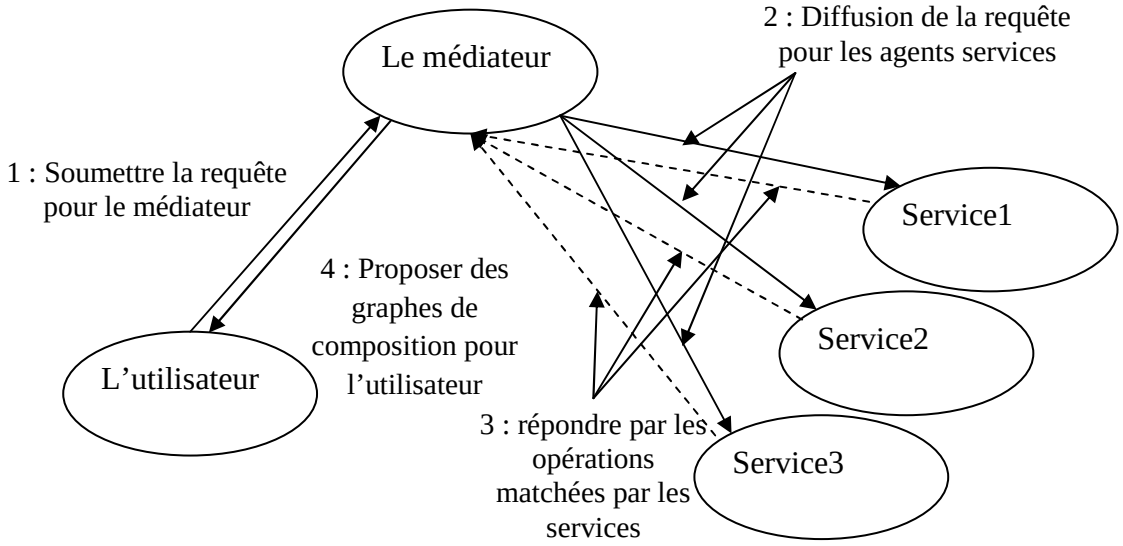


Figure V. 1. Les différentes interactions entre les différents agents

Le protocole de communication, comme il est illustré dans la figure précédente, est exécuté au niveau des trois différents agents, l'agent utilisateur, l'agent médiateur et les agents services. La communication est initialisée par l'agent utilisateur qui initialise la composition par l'envoi des paramètres de la requête à l'agent médiateur qui va lui-même diffuser aux agents services les fonctionnalités de la requête et attend les réponses de ces derniers. A la réception du message du médiateur par les agents services, ils effectuent le matching et renvoient le résultat à l'agent médiateur, à la réception des résultats de ce dernier, il effectue la composition, et finalise la communication par l'envoie du plan de composition à l'agent utilisateur. La communication est faite par l'ordre indiqué.

Dans cette approche, nous ne nous sommes pas intéressés aux problèmes tels que le problème d'interblocage, perte de messages, identification mutuelle des agents…etc. nous nous sommes intéressés plutôt aux processus du matching et de la composition.

V.3. Généralités

Un système multi agents est un système distribué où les agents peuvent communiquer entre eux. Nous allons donc implémenter notre approche dans un environnement distribué. Le calcul du plan de composition s'effectue au niveau du médiateur.

D'une manière générale, le *matching* des services web dépend de trois éléments importants :

1- La description des services,

2- La description de la tâche à réaliser ou la requête à satisfaire.

3- L'algorithme de matching.

Nous rappelons les définitions données au chapitre III,

– La description d'un service web est constituée des différents contextes du comportement de service **(**$Ctxt^{Pre}$,$Ctxt^{Desc}$**,** $Ctxt^{Post}$**).**

– La description de la tâche ou la requête est extraite de la même façon de celle du service.

– L'algorithme de *matching* est chargé de mettre en correspondance la description des services avec la description de la tâche à réaliser par la substitution des deux comportements précédents.

V.4. Le matching

La correspondance entre les services web et la tâche à réaliser peut s'effectuer à plusieurs niveaux. Elle peut s'effectuer au niveau des entrées/sorties d'un service ou au niveau des conversations d'un service.

V.4.1. Le matching au niveau des entrées/sorties d'un service

Le *matching* au niveau des entrées/sorties d'un service correspond à la recherche d'un service qui nécessite certaines données en entrée et qui retourne certaines données en sortie. Dans cette catégorie d'algorithmes, les informations considérées dans la description des services sont ses entrées/sorties offertes par le service. La requête est également spécifiée sous forme d'un ensemble d'entrée/sorties requises.

V.4.2. Le matching basé sur les conversations

Dans cette catégorie d'algorithmes de *matching*, on suppose que la description de la conversation d'un service web offre plus de détails sur le fonctionnement du service que de prendre seulement les conditions d'entrée et de sortie.

V.5. Description de notre approche

Notre approche est basée sur le principe de cette dernière catégorie du matching. Les conversations se font entre les séquences d'opérations du comportement du service et de l'utilisateur en passant par les états du comportement (les détails sont donnés dans la suite de cette section).

Dans notre approche, la requête de l'utilisateur est considérée comme un processus abstrait (sans aucune référence concrète à un service particulier). Ce processus est constitué d'une séquence d'opérations abstraites qui transforment les précontraintes de la requête de l'utilisateur à ses postcontraintes, en générant des contraintes intermédiaires (describing-constraints). Chaque opération possède ses précontraintes et postcontraintes qui apparaîtront après l'exécution de l'opération.

L'ensemble des contraintes entre deux opérations nous donne un état du comportement qui est défini par le tuple des contraintes suivantes $<f^e, f^i>$. L'ensemble des états du comportement nous donnent le modèle du comportement de l'utilisateur.

Nous supposons que le processus utilisateur est décomposable en opérations élémentaires. L'ensemble des opérations élémentaires constitue le comportement interne de la requête.

Définition 13

Soit un modèle du comportement utilisateur β_u, l'opération élémentaire $ElmOp_i$ est la transformation de l'ensemble de contrainte au point (*i-1*) à l'ensemble de contraintes du point *i* ; elle est définie comme étant le triplet $<Op_i, S_{i-1}, S_i>$ tel que S_i *et* S_{i-1} sont les états du comportement aux points *i et i-1* respectivement.

La figure suivante illustre l'opération élémentaire,

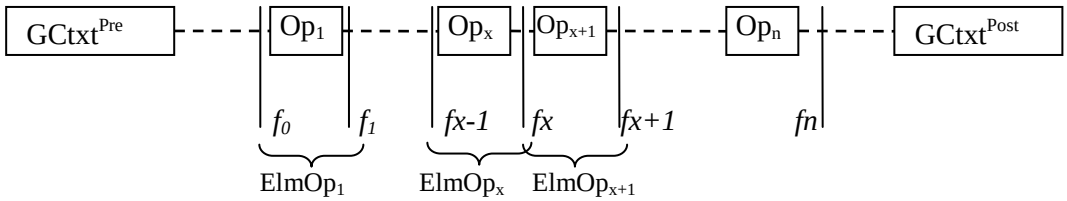


Figure V. 2. Les différentes opérations élémentaires

La conversation est définie comme suit :

Définition 14

Soit deux opérations élémentaires successives, $ElmOp_i$ et $ElmOp_{i+1}$. La conversation entre ces deux opérations est l'ensemble f_x de contraintes au point de transition *x* entre les deux opérations qui représente l'état $S_x <f_x^e, f_x^i>$. (Dénoté $Conv_x$)

Autrement dit, pour que deux opérations $ElmOp_i$ et $ElmOp_{i+1}$ s'exécutent en séquence, il faut que les contraintes effectives au point *x* (f_x^e)soient satisfaites.

Définition 15

Pour qu'une conversation $Conv_x$ soit correctement exécutée, l'ensemble des contraintesf_x^e au point *x* doivent être satisfaites.

Par la suite, notre approche suppose les services comme étant une séquence d'opérations élémentaire définies de la même façon que celles de la requête. L'ensemble des opérations élémentaires sont considérées indépendantes les unes des autres formant ainsi le processus du service qui est composé de processus atomiques représentant l'opération élémentaire.

Définition 16

Un processus atomique est la fonction de la transformation (le passage) d'un ensemble de contraintes à un autre ensemble de contraintes ; ceci se fait en satisfaisant le premier ensemble de contraintes.

V.6. L'algorithme de matching

Le but dans cette approche est de trouver un ensemble de services qui peuvent satisfaire la requête de l'utilisateur si un seul service ne suffit pas. Pour cela, le processus du matching

essaye de trouver un matching d'un fragment dans la séquence d'état du processus de l'utilisateur au niveau de chaque agent « service », ensuite rassembler l'ensemble des fragments pour construire une composition qui permettra de réaliser la matching de tout le processus de la requête.

Définition 17

Un fragment (*chunk*) est une opération élémentaire ou un ensemble d'opérations élémentaires successives dans la séquence d'opérations de la requête qui est matchée par une séquence d'opérations de service.

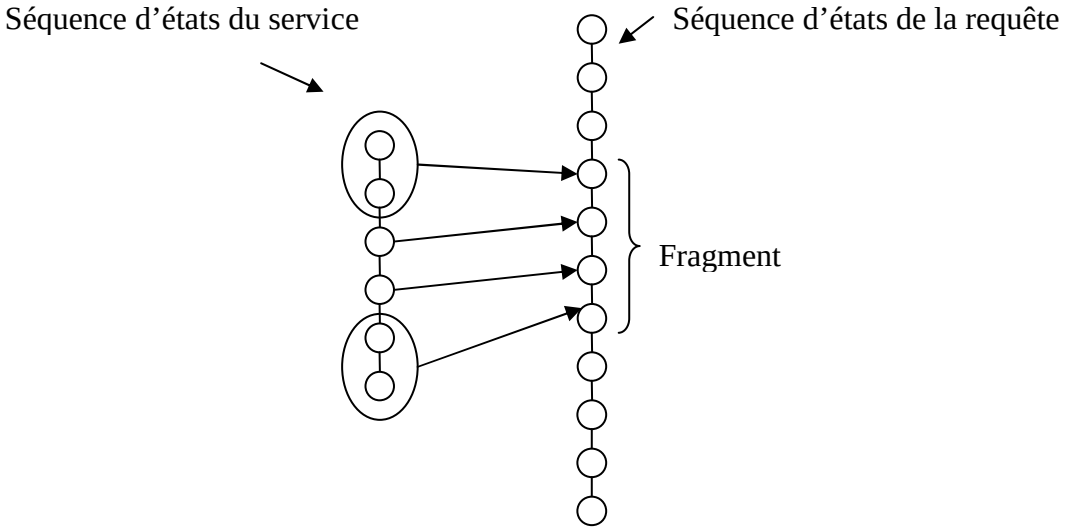


Figure V. 3. La représentation d'un fragment

Le principe de l'algorithme de matching proposé est d'effectuer un matching élémentaire (matching de chaque opération élémentaire) par chaque agent *service* pour la requête et d'envoyer le résultat au médiateur qui compose les différents services participant au matching de la requête.

V.6.1. Le matching de deux opérations élémentaires

Proposition 1

Soit deux opérations élémentaires de service et de la requête respectivement, $ElmOp_{(i,s)}$ et $ElmOp_{(j,t)}$, $ElmOp_{(i,s)}$ matche $ElmOp_{(j,t)}$ (dénoté $ElmOp_{(i,s)} \trianglerighteq ElmOp_{(j,t)}$) si $f^e_{(i-1,s)} \trianglerighteq f^e_{(i-1,t)}$ et $f^e_{(i,s)} \trianglerighteq f^e_{(i,t)}$.

Nous définissons la composition de deux opérations élémentaires de deux services différents par :

Proposition 2

Soit deux opérations élémentaires de deux services S_1 et S_2 respectivement, $ElmOp_{(i,1)}$ et $ElmOp_{(j,2)}$, qui matchent respectivement deux opérations élémentaires successives d'une requête, $ElmOp_{(req,k)}$ et $ElmOp_{(req,k+1)}$,

$ElmOp_{(i,1)}$ peut réaliser une composition avec $ElmOp_{(j,2)}$ (dénoté $ElmOp_{(i,1)} \prec ElmOp_{(j,2)}$), si $f_{(i,1)} \trianglerighteq f^{e}_{(j-1,2)}$.

La proposition précédente nous permet de montrer que la condition pour composer deux processus atomiques, leur conversation doit être correctement exécutée, c'est que les postcontraintes du premier processus doivent satisfaire les précontrainte du deuxième processus. Autrement dit, le principe de la composition entre deux services est maintenu (les postcontraintes du premier service doivent satisfaire les précontrainte du deuxième service).

V.6.2. Le déroulement de l'algorithme

Comme nous avons souligné auparavant, notre approche est basée sur les capacités d'agents cognitifs, où chaque agent est responsable de réaliser sa tache. Il y a trois types d'agents :

- l'agent utilisateur : Cet agent initie la composition, il émet une requête au médiateur constituée des différentes spécifications de l'utilisateur, les précontraintes et les postcontraintes de la requête. Ainsi qu'une référence sur l'ontologie domaine utilisé.

- L'agent médiateur : il connaît au préalable la localisation des agents fournisseurs de services, ainsi que leurs ontologies de domaine d'application. Son rôle est d'extraire la sémantique de la requête en correspondance à l'ontologie du domaine d'application utilisée, et de redistribuer la requête aux agents *services* concernés (qui ont le même domaine ontologique), et de récupérer les réponses (services offerts) avant de les envoyer au client. Au niveau de cet agent, toutes les informations sur la sémantique et les spécifications fonctionnelles de la requête sont sauvegardées. Cet agent est responsable de réaliser un bon plan de composition des services retournés et de le remettre à l'agent utilisateur.

- Les agents fournisseurs : Ces agents reçoivent la requête de l'agent médiateur qui est sous forme d'une séquence d'états (qui forment l'état du comportement de la requête). Ensuite, ces agents réalisent un matching direct avec leurs services.

Après avoir terminé le matching d'états, les agents fournisseurs déterminent les segments matchés de la requête, puis ils les envoient à l'agent médiateur qui va essayer de composer les différents segments afin de reconstruire le processus de la requête. Si l'opération a réussi, il détermine le bon plan de composition des services participants et le renvoie à l'utilisateur, sinon il renvoie un message pour le client lui confirmant qu'il n'existe pas de plan de composition qui satisfait toute la requête.

V.6.3. Le pseudo algorithme de la composition

Au niveau du client

Algorithme 4 : l'algorithme de la composition au niveau de l'agent client

Début

Initier la composition.
Déterminer les *paramètres* de la requête (préconditions, postconditions, fonctionnalités, ontologie)
Envoyer *requête (paramètres, médiateur, client$_i$).* //envoyer la requête au médiateur
Attendre (plan_composition, médiateur) // Attendre la plan de composition
Recevoir (plan_composition, médiateur, client$_i$)
FIN

L'algorithme 4 s'exécute au niveau de l'agent client qui initie la composition en envoyant *requête (paramètres, médiateur, client$_i$)* au médiateur qui est constituée des paramètres de la requête, l'adresse du médiateur et l'identifiant de client (référence de l'application qui demande le service).

Avant de donner l'algorithme dont l'exécution se fait au niveau du médiateur et l'agent fournisseur du service, nous donnons quelques définitions essentielles utilisées dans les deux algorithmes.

ElmOp_Table[] est un vecteur calculé par chaque agent *service,* sa taille est égale au nombre d'opérations élémentaires de la requête qui est m-1 (m est le nombre d'états dans le comportement de la requête),

Chaque élément du vecteur est constitué de quatre champs : *matched,* est une valeur booléenne qui indique l'état du matching de l'opération élémentaire impliquée, *begin_cluster* : indique l'état de début du cluster qui matche cette opération dans le cas où *matched=1, end_cluster* : indique l'état de début du cluster, *ref_serv* : est la référence du service impliqué (elle peut être l'URL du service)

Nous avons utilisé les automates d'états finis pour réaliser la composition des services, ainsi que la notion d'état orphelin,

Définition 18

Un *état orphelin* est un état inaccessible ou qui ne mène pas à l'état final. En d'autre terme, un état orphelin est un état qui ne possède pas soit un arc entrant, soit un arc sortant, soit les deux en même temps.

L'état initial et l'état final ne sont pas des états orphelins.

L'algorithme 5 et 6 suivants sont exécutés respectivement, au niveau du médiateur et l'agent fournisseur.

Au niveau du médiateur

Algorithme 5 : l'algorithme de la composition au niveau de l'agent médiateur

Entrée n le nombre des agents_services S
Début
Attendre *requête (paramètres, médiateur, client)*
// A la réception de la requête du $client_i$
Recevoir *requête (paramètres, médiateur, $client_i$)*
Déterminer le modèle du comportement $\beta_j$de la requête.
Déterminer GCtxt($GCtxt^{pre}$, $GCtxt^{post}$, $GCtxt^{desc}$)

m = Déterminer le nombre d'état de la requête

Pour chaque *agent_service* S_i enregistré dans la base du médiateur **faire**
    Envoyer *requête (GCtxt, β_j, G, S_i, médiateur)*
Fin pour
Pour tout i allant de 1 à n **faire**
    Attendre *réponse* (*ElmOp_Table[i]*, médiateur, S_i) du chaque agent_service S_i.
Fin pour

// A la réception de la réponse de service S_i
Pour chaque agent_service S_i **faire**
Recevoir *réponse* (*ElmOp_Table*[i], médiateur, S_i)
Fin pour

// Générer un ensemble d'états formant l'automate de composition
Générer m états_automate // m est le nombre de l'automate de composition.
Mettre (T_0 l'état initial et $T_{(m-1)}$ l'état final)
Pour chaque S_i **faire**
    Pour i allant de 1 à m-1 **faire**
        Si *ElmOp_Table*[i].matched=1 **alors**
        j= i
        Tantque (j ≤ m-1 et *ElmOp_Table*[i].matched=1)**faire**
            j=j+1
        Fin tantque
Créer ARC de (i-1 à j)
Identifier l'arc par ref_serv et le cluster qui matche l'ensemble des opérations élémentaire
        i = j
    Fin pour
Sauvegarder l'automate.
Supprimer les états orphelins.
Créer plan_composition
Si (∄ de chemin de l'état initial à l'état final **alors**
Envoyer (pas de plan de composition**)**
Fin si

Envoyer (plan_composition, médiateur, $client_i$)
FIN

Au niveau de l'agent service Si

Algorithme 6 : l'algorithme de la composition au niveau de l'agent fournisseur

Entrée : « n » : le nombre d'états source,
Un tableau d'états source (S).
β_i : Le modèle de comportement de service.

Début
Variable *ElmOp_Table {begin_cluster, end_cluster}*
Attendre *requête (GCtxt, β_j, G, S_i, médiateur)*
// à la réception de la requête du client de la part du médiateur
Recevoir *requête (GCtxt, β_j, G, S_i, médiateur)*

m = Déterminer le nombre d'état de la requête
T = Déterminer le tableau d'états de la requête
// Appliquer l'algorithme 1 pour la requête de l'utilisateur avec le comportement du service.

Algorithme1 (n, m, S, T, β_i, β_j)
Récupérer Table_cluster,

// Générer la table du matching des opérations élémentaires, pour chaque service avec le //processus de la requête, une séquence de m états, possède (m-1) ElmOp.
ElmOp_Table {matched, begin_cluster, end_cluster, ref_serv}
Créer *ElmOp_Table : tableau [1 … m-1]*
Initialiser *ElmOp_Table[].matched=0* *ElmOp_Table[].ref_serv=S_i*
Pour j allant de 0 à m-1 **faire**
Si (j +1 ≤ m-1) **alors**
Si ((T_j et T_{j+1}) ∉ unmatched) **alors**
ElmOp_Table[j+1].matched =1,
ElmOp_Table[j+1].begin_cluster= Table_Cluster[j].Begin_cluster
ElmOp_Table[j+1].end_cluster= Table_Cluster[j+1].End_cluster
Fin si
Fin si
Fin pour

Envoyer *réponse* (*ElmOp_Table*[], médiateur, S_i) ;

FIN

Nous supposons dans nos travaux que l'agent *fournisseur* (*service*) est un agent qui possède un seul service. Nous pouvons faire une extension pour l'application de telle sorte que l'agent *fournisseur* peut avoir un ou plusieurs services. Dans ce cas, nous exigeons de l'agent *fournisseur* d'effectuer le matching pour chaque service offert indépendamment et de construire les tables de matching des opérations élémentaires de chaque service, puis les envoie au médiateur qui va réaliser la composition avec les différents services.

Définition 19

Un automate de composition est un automate d'états finis, les états de l'automate représentent les états du comportement de la requête, les arcs de l'automate représentent le cluster de service qui matche les états de comportement de la requête de l'état qui représente le début d'arc jusqu'à l'état qui représente la fin d'arc.
Nous comprenons ici, que la composition se fait au niveau de l'état du comportement de la requête qui représente la fin du premier arc et le début de deuxième arc. Ce qui veut dire que la composition se fait entre la dernière opération élémentaire du premier service et la première du deuxième service, si la proposition 2 est vérifiée.

V.6.4. La complexité de l'algorithme

La complexité de l'algorithme est de $O(m.\frac{n^3}{2})$.

La complexité réside toujours dans la détermination de l'état du matching d'un état du comportement de la requête. Dans notre approche, le matching s'effectue au niveau de l'agent service, chaque agent effectue le matching indépendamment des autres agents. La complexité au niveau de chaque agent est égale à la complexité de l'algorithme 1, si nous supposons que n est le nombre d'états au niveau de la requête et m est le nombre d'agents service, alors nous trouvons que la complexité au niveau d'un agent est égale à $O(\frac{n^3}{2})$. La complexité pour l'algorithme est $O(m.\frac{n^3}{2})$.
Si nous considérons que l'environnement est distribué, les agents services travaillent en parallèle, ça nous donnera une complexité dans le temps est égale approximativement à $O(\frac{n^3}{2})$.
La complexité de la construction de l'automate de composition réside au parcours de la table du matching des opérations élémentaire *ElmOp_Table* , ce qui nous donne une complexité de *O(m.(n-1))*.

V.6.4. Un exemple de l'application

La figure suivante nous montre différents modèles de comportements, nous avons une séquence d'états qui représentent le modèle de comportement utilisateur (requête) et sept séquences d'états qui représentent des comportements des services web, nous essayons de trouver les différents plans de composition pour ces services web afin de satisfaire la requête de l'utilisateur en appliquant notre algorithme de la composition sur ces comportements. Nous trouvons les résultats suivants :

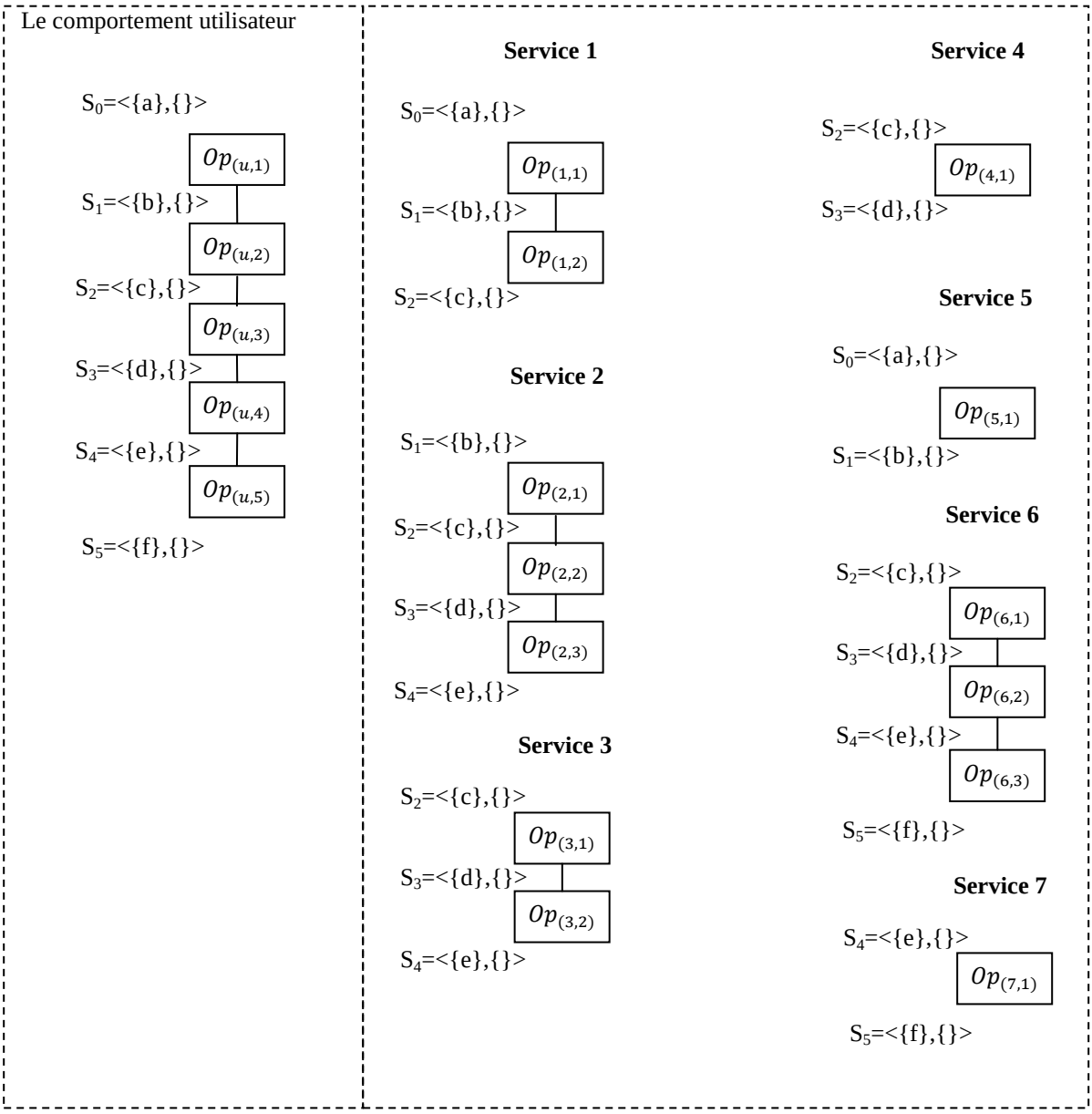


Figure V. 4. Le comportement de la requête demandé et les comportements des services

Dans l'exemple de la figure précédente, nous avons une séquence d'états de but constituée de six états (S_0 à S_5) avec cinq opérations, ce qui nous donne cinq opérations élémentaires ($ElmOp_1$ à $ElmOp_5$) selon la définition 13. Dans le coté droit de la figure, nous avons sept modèles de comportement de différentes tailles. Notre algorithme est appliqué en effectuant le matching de chacune de ces séquences indépendamment les unes des autres.
L'algorithme considère trois types d'agents :

- L'agent utilisateur exécute l'algorithme 4 qui initie la composition. Après la détermination des différents paramètres de la requête, il les envoie pour l'agent médiateur, ensuite il attend la réponse du médiateur.
- L'agent médiateur exécute l'algorithme 5, après la réception des différents paramètres de l'agent utilisateur il détermine le modèle de comportement de la requête et ses différents contextes puis il les diffuse pour chaque agent service S_i enregistré dans sa base et il attend la réponse;

- Chaque agent service exécute l'algorithme 6, après la réception de la requête du médiateur, il détermine le nombre d'états de la requête. Il exécute ensuite l'algorithme 1 principal sur la séquence d'états de la requête.

L'algorithme 1 retourne la *table_cluster*. A partir de cette dernière, l'agent service calcule la table du matching des opérations élémentaires (*ElmOp_Table*) du service avec la séquence d'état de la requête. L'opération élémentaire est constituée de deux états consécutifs avec l'opération qui transforme le premier état au deuxième état.

Après que la table *ElmOp_Table* soit calculée, elle est envoyée au médiateur qui se charge de la composition.

Lorsque le médiateur reçoit les tables *ElmOp_Table* de chaque service, il continue l'exécution de l'algorithme 5 et suit les étapes suivantes :
Premièrement, il construit la table *ElmOp_Table* globale de tous les services, il obtient *ElmOp_Table* suivante:

	ElmOp 1	*ElmOp 2*	*ElmOp 3*	*ElmOp 4*	*ElmOp 5*
Service 1	1	1	0	0	0
Service 2	0	1	1	1	0
Service 3	0	0	1	1	0
Service 4	0	0	1	0	0
Service 5	1	0	0	0	0
Service 6	0	0	1	1	1
Service 7	0	0	0	0	1

Si nous prenons l'exemple du service1, nous trouvons que service1 matche les trois premiers états de la requête ce qui veut dire qu'il matche les deux premières opérations élémentaires. Alors l'algorithme met un 1 dans le champ *matched* pour chacune des opérations élémentaires *ElmOp1* et *ElmOp2*. Ainsi de suite pour les autres services.
A partir de cette table, l'agent médiateur construit l'automate de composition suivant :
Tout d'abord, le médiateur crée 6 états de l'automate. Il met l'état 0 comme état initial et l'état 5 comme état final, puis il parcourt la table précédente pour former l'automate de composition. Il commence par le service1, il trouve que *matched* pour *ElmOp1* et *ElmOp2* est égal à 1, alors il crée un arc commençant du premier état de *ElmOp1* jusqu'au dernier état de *ElmOp2*, c'est-à-dire de l'état S_0 à S_2. Pour le service2, il crée un arc de l'état S_1 à S_4 et ainsi de suite.
A la fin du parcourt de la table de matching, nous obtenons l'automate représenté dans la figure suivante :

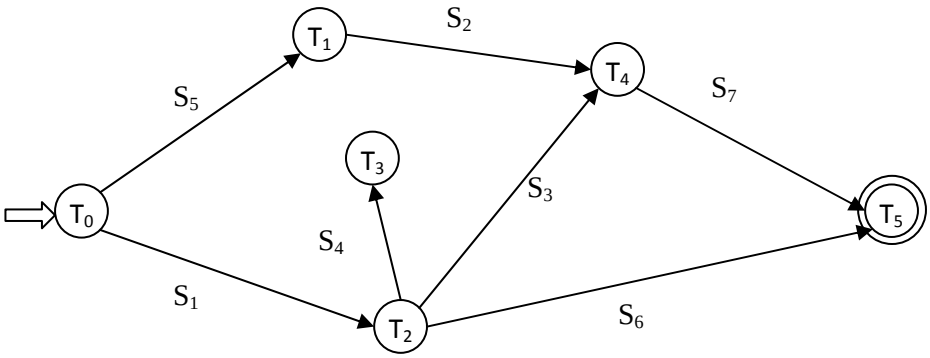


Figure V. 5. L'automate de composition

A la fin de la construction de l'automate, nous remarquons que T_3 est un état orphelin, alors algorithme 5 procède à l'élimination des états orphelins. Nous obtiendrons l'automate suivant :

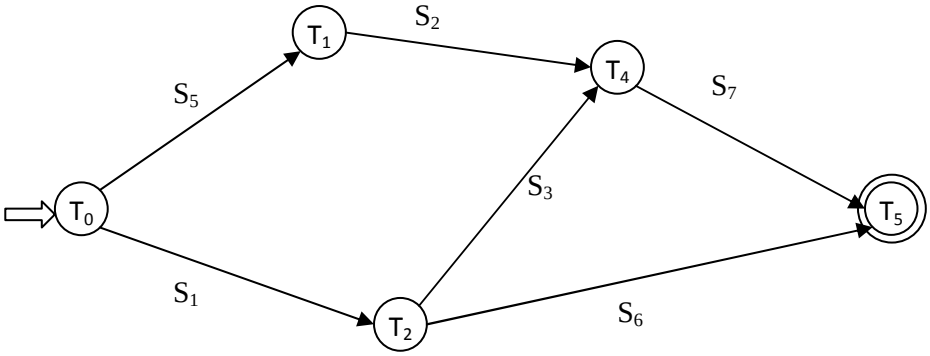


Figure V. 6. Le plan de composition

L'existence d'un chemin de l'état 0 à l'état 5 veut dire qu'une composition est trouvée. Les services participants sont les services indiqués sur les arcs du chemin.
A partir de ce plan de composition, nous aurons les différentes compositions de service. Qui sont (*service1, service6*), (*service1, service3, service7*), (*service5, service2, service7*). Ce plan peut être filtré avec d'autres paramètres (spécification non fonctionnelles, spécifications non techniques etc..), par exemple ; les paramètres de la qualité de service, le prix etc.…

Enfin ce plan est envoyé à l'agent utilisateur.
Si nous comparons l'exemple précédent, notre algorithme avec celui proposé par Tari [59] qui réalise une agrégation séquentielle des services web, nous trouvons une différence importante au niveau de la complexité.
Le *GAP agregator* proposé par Tari utilise une structure de donnée qui facilite l'agrégation. Cette structure utilise une liste des chunks en tableau de piles où chaque pile contient les chunks ayant la même valeur Start. Pour chaque état, il existe une pile de Chunks qui commence avec son indice, comme indiqué dans la figure suivante [59].

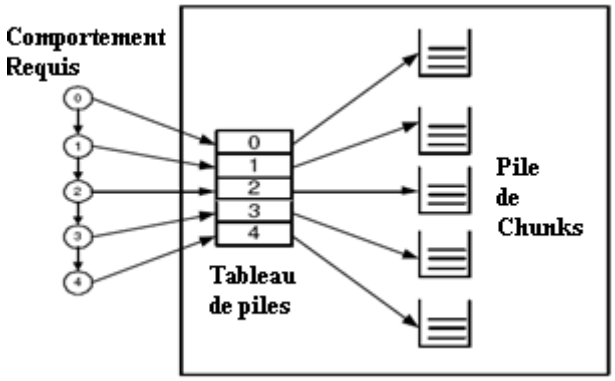


Figure V. 7. Tableau de piles de chunks

En appliquant l'algorithme sur notre exemple, le processus commence par parcourir le service d'enregistrement de services. Il commence par vérifier les chunks dans la pile correspondant au premier état cible S_0, noté Pile0. Pour chaque chunk dans la Pile0, on crée une nouvelle pile contenant tous les chunks qui commencent avec la valeur de son champ End.

A la fin de l'exécution, La liste correspondante dans le tableau de piles serait:

[Pile 0] contient Chunk_1 = <Service1, 0, 5>, Chunk_6 = < Service5, 0, 5>.

[Pile 1] contient Chunk_2 = <Service3, 2, 5>, Chunk_4 = <Service6, 2, 5>, Chunk_5 = <Service4, 2, 5>.

[Pile 2] contient Chunk_3 = <Service7, 4, 5>.

[Pile 3] contient Chunk_2 = <Service2, 1, 5>.

[Pile 4] contient Chunk_8 = <Service7, 4, 5>.

L'algorithme est initialement invoqué par (requête, 0, 5), il construit pile0 qui contient chunk_1 et chunk_6, commençant par chunk_1, l'algorithme trouve que (service1) ne couvre pas toute la séquence de la requête, il fait donc un appel récursif avec (requête, 2,5) ; cet appel construit pile1 qui contient chunk_2, chunk_4, chunk_5. Il commence par chunk_2 et trouve que la composition (service1-service3) ne couvre pas toute la séquence de la requête, d'où il fait un appel récursif avec (requête, 4, 5) et construit pile2 qui contient chunk_3. L'algorithme trouve que la composition (service1-service3-service7) matche toute la séquence de la requête, alors il retourne la composition comme solution. Ensuite, il fait un retour arrière à son précèdent ancestral chunk_4, il trouve la composition (service1-service6) est une solution. Il procède à chunk_5 qui n'est pas une solution, alors il fait un appel récursif avec (requête, 3, 5) et ne trouve aucun service qui peut être composé avec la composition (service1-service4), alors l'algorithme élimine chunk_5 de la liste des piles.

Il fait un retour arrière, et il procède à chunk_6 ; il trouve que service5 ne couvre pas toute la séquence, alors il fait un appel récursif avec (requête, 1, 5) ; il construit pile3 qui contient chunk_7, il trouve que la séquence (service5-service2) ne couvre pas toute la séquence, d'où il fait un appel récursif avec (requête, 4, 5) et il construit pile4 qui contient chunk_8. Finalement il trouve que la composition (service5-service2-service7) est une solution ; alors il termine par retourner les trois solutions (*service1, service6*), (*service1, service3, service7*), (*service5, service2, service7*) ; comme il est indiqué dans la figure suivante :

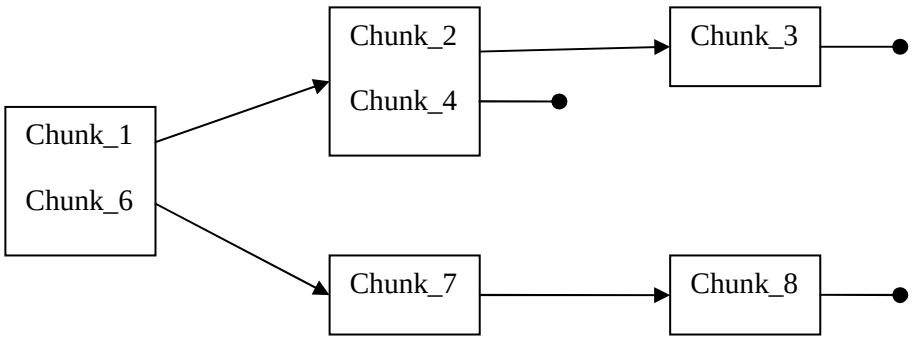


Figure V. 8. La trace du processus d'agrégation

Nous disons que l'algorithme proposé par Tari donne les mêmes résultats que notre algorithme. Nous remarquons qu'il existe des cas où l'algorithme séquentiel proposé par Tari peut effectuer des matching et trouve que le cas n'aboutie à aucune solution comme le cas du chunk_5, où la composition des deux services (service1-service4) ne peut être composée avec aucun service, d'où il fait des appels récursifs. Ce cas est déterminé dans notre algorithme par les états orphelins qui sont supprimés sans complexité dans le temps ni dans les instructions contrairement à celui proposé par Tari qui nécessite un matching de toute la séquence des services composés. Ce qui augmente énormément la complexité de l'algorithme.

Si on calcule la complexité des deux algorithmes pour cet exemple, nous trouvons :

Pour le cas de l'algorithme de Tari, elle est égale à n^m [59]; m étant le nombre d'états dans la séquence de la requête, et n est le nombre de services ; sachant que chaque vérification d'un matching pour une séquence d'état d'un ou plusieurs services composés est de complexité de $O(m^4)$, ce qui nous donne la complexité totale de $O(n^m\ m^4)$ qui est exponentielle, alors que notre algorithme est de complexité de $O(n.\frac{m^3}{2})$ et la complexité de la construction de l'automate de composition est de $O(n.(m-1))$. Alors nous concluons que la complexité de notre algorithme est inférieure considérablement à celle du premier algorithme.

V.7. conclusion

Ce dernier chapitre est consacré au développement de l'approche de composition que nous avons proposée. Notre approche est une approche de composition dynamique des services web sémantiques. Nous avons utilisé un système multi-agents. Nous avons proposé un protocole d'interaction qui regroupe trois types d'agents : l'agent client, l'agent médiateur et les agents fournisseurs. L'architecture proposée présente beaucoup de souplesse, en matière d'ajout, de retrait et de modification d'agents fournisseurs. Cet aspect nous donne tous les avantages des implémentations modulaires où chaque module est indépendant de l'autre. Le matching s'effectue au niveau des agents fournisseurs qui essayent de trouver un matching pour des segments du comportement de la requête du client. La composition s'effectue au niveau de l'agent médiateur, qui lui-même utilise les automates d'états finis pour la réalisation de sa tâche.

L'approche de composition proposée prend les avantages des deux approches suivantes : les applications distribués, cela en implémentant les agents *fournisseurs* d'une façon distribuée ce qui va nous permettre d'avoir une complexité dans le temps plus réduite, et les *brokering approches* avec qui l'environnement de l'exécution est optimisé et cela en gardant la trace de l'exécution.

CONCLUSION GENERALE

Conclusion Générale

Le travail effectué dans le cadre de ce projet constitue une phase incontournable pour découvrir le domaine de l'internet et les services web. Un des objectifs majeurs que nous nous sommes fixés au départ, était d'explorer ce domaine, tout en se focalisant sur les problèmes qui y sont posé, notamment le problème de la découverte des services web, en se basant sur la sémantique des services et de la requête, qui s'impose aujourd'hui en tant que challenge.

Nous avons étudié les différents obstacles pour l'automatisation du service de matching des services web et les différents algorithmes existants, puis nous avons proposé un algorithme de matching avec une complexité inferieure à celle des algorithmes existants. Notre deuxième contribution réside à la proposition d'une approche de composition qui résout les problèmes des approches d'agrégation séquentielles.

Le premier problème auquel nous nous sommes intéressés est celui du matching des services web. Pour automatiser le processus du matching des services web, le matchmaker doit être capable de déterminer la justesse des résultats du matching en respectant les buts de l'utilisateur définis dans sa requête. Alors, capturer la sémantique des services, utilisateurs et le domaine d'application devient une nécessité afin de rendre le matchmaker capable d'analyser et de comparer ces sémantiques. Contrairement aux approches de matching syntaxiques existantes qui sont basées uniquement sur les aspects fonctionnels des services web, les approches qui se basent sur la représentation des services basée sur leurs comportements intégrant les spécifications fonctionnelles de haut niveau et de bas niveau d'abstraction, est plus appropriée.

Dans ces dernières approches, nous trouvons le travail proposé par Tari et al. [59] est le plus intéressant, qui donne de meilleurs résultats comparant aux autres techniques existantes. Alors nous nous sommes basés sur l'objectif de l'améliorer en réduisant sa complexité, ceci en lui intégrant des heuristiques et des théorèmes qui permettent la flexibilité et la performance de cette technique. Tari a proposé un modèle service qui permet la représentation formelle des services. Cette représentation intègre les différents types des spécifications fonctionnelles, cela pour renforcer la découverte sémantique de service. Le haut niveau permet la capture des

comportements externes et les contextes de réalisation des objectifs ainsi que les rôles du service. Le bas niveau d'abstraction permet la capture des informations sur les accès aux interfaces, de la bande passante ainsi que sur les protocoles de communication. Nous ajoutons aussi que ce modèle permet la composition de plusieurs services pour former un seul service virtuel qui satisfait la requête si ces derniers ne le permettent pas individuellement

Dans cette première partie de notre travail, nous avons proposé un algorithme de matching des spécifications de haut niveau (rôle, but, contextes et le comportement externe) afin de les utiliser dans le processus de médiation pour l'amélioration du niveau ontologique des domaines d'applications, notre algorithme utilise le modèle G+ et le concept des scénarios pour capturer le comportement externe des services qui sera plus utile de l'appliquer dans les approches de composition m-to-n. Nous avons défini le type de la sémantique capturée, les règles du matching appliquées, les modèles utilisés, le critère de la satisfaction de l'objectif etc. l'algorithme repose sur le schéma SWSMF (*Semantic Web Services Matching Framework*) pour capturer les spécifications techniques et non techniques des services web. Le concept du matching de notre algorithme est le principe de substitution de contraintes car les fonctionnalités demandées par l'utilisateur sont représentées par un ensemble de contraintes et le matchmaker doit trouver le service qui satisfait ces contraintes, ce qui traduit le problème de matching en problème de satisfaction de contraintes.

Notre contribution utilise des structures de données qui sont utilisées pour sauvegarder l'état du matching pour les états objectif avec tous les clusters possibles qu'on peut former à partir de la séquence de la source. Ces structures permettent d'avoir l'état du matching des clusters formés d'un état source étendu avec ses prédécesseurs (*reverse expansion*) obtenu en faisant étendre le premier état du cluster avec ses successeurs (*down expansion*). Cette technique nous permettent de réduire énormément la complexité par rapport à l'algorithme proposé par Tari.

Ces structures nous permettent aussi de récupérer des segments (*chunk*) dans la liste des états but qui sont matchés par la liste des états sources dans le cas où la séquence des états source ne peut pas matcher toute la séquence des états but. Ce qui nous permet d'appliquer l'algorithme sur différentes séquences source indépendamment et de récupérer les différents segments matchés par chaque séquence de la source. Cette caractéristique nous donne une idée de l'utiliser dans une approche de composition concurrente.

En utilisant les extensions d'états, nous considérons notre technique appartenant à la catégorie des approches m-to-n, l'avantage ne s'arrête pas ici, grâce aux structures utilisées et au principe de l'extension des états, notre algorithme peut être utilisé pour des séquences de même taille égales c'est-à-dire une technique de matching n-to-n.

Le deuxième problème auquel nous nous sommes intéressés, c'est le problème de la composition de services. Les approches d'agrégation séquentielles manquent de flexibilité et de performance car elles utilisent des tests de matching pour des segments (*chunks*) dans la séquence des états but par des appels récursif, ce qui donne une complexité exponentielle égale au nombre de services puissance le nombre d'états dans la séquence de l'objectif. Notre approche de composition est dynamique, elle utilise différents types d'agents afin de permettre la décentralisation du processus du matching ces différents agents services (fournisseurs) exécutent l'algorithme indépendamment les uns des autres. Le processus de la composition s'effectue d'une façon centralisée au niveau d'un seul agent qui s'appelle le médiateur ;. Notre approche utilise aussi quelques structures de données et les automates d'états finis dans le processus de la composition au lieu d'utiliser la représentation par arborescence, cela pour permettre de réduire la complexité et de la rendre linéaire.

A partir de toutes ces caractéristiques, nous disons que notre approche nous donnera des meilleurs résultats par rapport aux approches séquentielles car elle prend les avantages des approches distribuées qui implémentent des algorithmes qui peuvent être exécutés en parallèle, ce qui réduit considérablement la complexité.

Un autre avantage pour notre approche, est qu'elle prend aussi les avantages des *brokering-appproaches* qui peuvent optimiser l'environnement de l'exécution en gardant la trace de l'exécution, car dans notre cas, l'agent médiateur est considéré comme broker qui reçoit la requête de l'utilisateur, puis il la redistribue pour les agents services qui vont effectuer la matching, ces derniers renvoient les résultats à l'agent médiateur qui va lui-même réaliser la composition et s'en charge d'envoyer le résultat à l'utilisateur.

Perspectives

Afin de compléter notre travail, nous envisageons les points suivant comme amélioration de travail proposé :

Une amélioration dans la flexibilité et les performances : Malgré que nos algorithmes ont réduit énormément la complexité, des travaux sont nécessaires pour l'optimisation de ces algorithmes proposés pour une meilleure performance et flexibilité et de réduire plus cette complexité en utilisant différentes heuristiques. Cette tâche est une direction importante pour les travaux futurs.

Le service de sélection : Notre contribution se base sur les spécifications fonctionnelles de haut niveau, alors dans une autre direction de recherche, le développement d'un schéma de matching pour les spécifications non fonctionnelles et non techniques et très important. Ceci exige de définir quelles sont les règles de matching et le critère de la satisfaction de l'objectif qui seront utilisés.

Le service de la composition : nous avons proposé une approche de composition qui réalise la composition d'une façon centralisée au niveau de l'agent médiateur, l'implémentation des algorithmes de composition en décentralisant le processus de composition et un point aussi important. Nous exigeons pour cela d'implémenter un protocole de communication qui permet d'éviter les interblocages et des verrous qui permettent de gérer cette composition au niveau de chaque agent et de permettre la mise à jour des registres des services continuellement.

L'implémentation des contributions : malgré que nous avons validé nos contributions par des exemples illustratifs qui nous donnent les résultats de nos algorithmes et les calculs qui sont faits et démontrent la valeur de la complexité, une implémentation et des tests restent indispensables pour valider ces résultats. Cette tâche à été donnée dans un sujet PFE (Projet de Fin d'Etude) pour la section des ingénieurs d'état en informatique, elle est en cours de réalisation.

Bibliographie

[1] S. Agarwal, S. Handschuh, S. Staab. Surfing the Service Web. 2003.
[2] A. Ankolenkar, M. Burstein, J.R. Hobbs, O. Lassila, D.L. Martin, S.A. McIlraith. DAML-S: Web Service Description for the Semantic Web. In *Proceedings of the First International Semantic Web Conference*, Sardinia, Italy, 2002.
[3] A. Ankolenkar, M. Burstein, et al. DAML-S: semantic markup for web services. 2002,
[4] V. Antonellis, M. Melchiori, B. Pernici, P. Plebani. A Methodology for e-Service Substitutability in a Virtual District Environment. In *Advanced Information Systems Engineering, 15th International Conference*, Austria, 2003.
[5] T. L. Bach. Construction d'un web sémantique multi-points de vue. PhD thesis, École des Mines de Nice à Sophia Antipolis, 2006.
[6] B. Benatallah, M.S. Hacid, C. Rey et F. Toumani. Semantic Reasoning for Web Services Discovery. *WWW Workshop on E-Services and the Semantic Web*, Budapest, Hungary, 2003.
[7] V. Benjamins, J. Contreras, O. Corcho. Six challenges for the semantic web. 2004.
[8] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini and M. Mecella. Automatic Composition of e-Services that Export their Behavior. Département d'informatique et système, université de Rome, Italie. 2005.
[9] T. Berners-Lee et E. Miller. The semantic web. présentation W3C, diapositive 17– Layer Cake, 2002. http://www.w3.org/Talks/2002/01/10-video/slide17-0.html.
[10] T. Berners-Lee. What the semantic web can represent, 1999. http://www.w3.org/DesignIssues/RDFnot.html
[11] A. Bernstein, M. Klein. Discovering Services: Towards High Precision Service Retrieval. Toronto, Canada, 2002.
[12] T. Bray, J. Paoli, C. Sperberg. Extensible Markup Langage (XML). W3C- Recom mendation, 1998. http://www.w3.org/TR/REC-xml.
[13] M. Burstein, A. Ankolenkar, M. Paolucci, et al. OWL-S: Semantic Markup for Web Services. http://www.daml.org/services/owl-s/1.0/owl-s.pdf
[14] C. Bussler. Semantic web services: reflections on web service mediation and composition, 2003.
[15] J. Cardoso. Workflow Quality of Service and Semantic Workflow Composition. *Ph.D. Dissertation. Department of Computer Science*, University of Georgia, Athens, 2002.
[16] F. casati, s. Ilnicki, L. Jin, V. Krishnamoorthy, M. Shan. Adaptive and dynamic service composition in eflow. In *Proceedings of the 12th International Conference on Advanced Information Systems Engineering (CAiSE)*, Stockhom, Sweden, 2000.
[17] J. Castillo, D. Trastour, C. Bartolini. Description logics for matchmaking of services. In *proceeding of workshop on application of description logics, Australia,* 2001.
[18] D. Chakraborty, F. Perich, S. Avancha, et A. Joshi. DReggie: Semantic Service Discovery for M-Commerce Applications. *In Workshop on Reliable and Secure Applications in Mobile Environment, 20th Symposium on Reliable Distributed Systems,* 2001.
[19] D. Connolly, F. V. Harmelen, I. Horrocks, D. L. McGuinness. DAML+OIL: Reference Description, Technical report, W3C : World Wide Web Consortium, http://www.w3.org/TR/2001/NOTE-daml+oil-reference-20011218, December, 2001.
[20] F. Curbera, M. Duftler et al. Unraveling the web Services web: An Introduction to SOAP, WSDL, and UDDI. 2002
[21] K. Decker, M. Williamson, K. Sycara. Matchmaking and brokering. In *proceedings of the 2nd international conference in multi-agent systems, japan*, 1996.
[22] I.M. Elgedawy, Correctness-Aware High-Level Functional Matching Approches For Semantic Web Services. PhD thesis, RMIT University, Melbourne, Australia. 2006.

[23] J. Ferber, les systèmes multi-agents : vers une intelligence collective. 1995.
[24] D. Fensel, C. Bussler. The web service modeling framework WSMF. Electronic Commerce Research and Application, 2002.
[25] D. Fensel, C. Bussler et A. Maedche. Semantic Web Enabled Web Services. In *International Semantic Web Conference, Sardinia*, Italy.
[26] T. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 1993.
[27] D. Harel, D.Kozen et J. Tiuryn. "Dynamic logic". Edition The MIT Press, 2000.
[28] R. Hamadi, B. Benatallah. A Petri Net-Based Model for Web Service Composition. School of Computer Science and Engineering, the University Of New South Wales, Sydney NSW 2052, Australia. 2003.
[29] J. Hendler, D. Nau, B. Parsia, E. Sirin, D. Wu. Automating DAML-S Web Services Composition Using SHOP2. 2003.
[30] I. Horrocks, F. Van Harmelen. Reference description of the DAML+OIL ontology markup langage. Draft report, 2001. http://www.daml.org/2000/12/reference.html.
[31] J. Javier Samper, F. Javier Adell, Leo van den Berg and J. José Martinez. Improving semantic web service discovery. Robotics Institute, University of Valencia, Valencia, Spain.
[32] Jena community, A Programmer's Introduction to RDQL, 2002, http://jena.sourceforge.net/tutorial/RDQL/index.html
[33] P. D. Karp, V. K. Chaudhri, J. Thomere. XOL: An XML-based ontology exchange language. 1999. http://www.ai.sri.com/pkarp/xol/xol.html.
[34] Y. Kalfoglou, M. Schorlemmer. Ontology mapping: the state of the art. The knowledge engineering review, 2003.
[35] P. Kellert, F. Toumani. Les web services sémantiques. Information – Interaction – Intelligence ; Revues en Science de traitement de l'Information, 2004.
[36] M. Klein, A. Bernstein. Towards High-Precision Service Retrieval. IEEE Internet Computing, 2004.
[37] G. Klyne, J. J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. Technical report, W3C : World Wide Web Consortium, http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/, February, 2004.
[38] S. Kouadri, A. Mostefaoui, B. Tafat-Bouzid. Using Context Information for Service Discovery and Composition, 2003.
[39] C. Lee, S. Helal. Context attributes: An approach to enable context-awareness for service discovery. In *proceedings of 3rd IEEE/IFSJ sumposium on applications and the internet*, Orlando, florida, USA, 2003.
[40] C. Lopez. Services Web et adaptabilité à l'utilisateur. Thèse de Master, Laboratoire LES-IMAG, Juin 2004.
[41] F. Mavaddat, S. V. Hashemian. A Graph-Based Approach to Web Services Composition. In *proceedings of the 2005 Symposium on Applications and the Internet (SAINT'05).* School of Computer Science, university of Waterloo. 2005.
[42] B. Medjahed, A. Bouguettaya et A. K. Elmagarmid. "A. Semantic web enabled composition of web services". *The VLDB Journal*, 12(4), pp. 333-351, November 2003
[43] T. Melliti. Interroperabilité des sevices Web complexes. Application aux systèmes multi agents. PhD thesis, université Paris IX Dauphine, décembre 2004.
[44] V. Monfort, H. Kadima. Les services web, techniques, démarches et outils. université Paris 11 ; 2003.
[45] W. Naylor, J. Padget, S.A. Ludwing, O. F. Rana. Agent-Based Matchmaking of Mathematical Web Services. Department of computer science, university of Bath, UK. 2005.
[46] OMG "*Object Management Group*", Catalog of Corba/IIOP specifications, 2004,

http://www.omg.org/technology/documents/corbaservices_spec_catalog.htm.
[47] OMG "*Object Management Group*", Catalog of Corba Facilities specifications, 2004, http://www.omg.org/technology/documents/corbafacilities_spec_catalog.htm.
[48] OSGi Alliance, OSGi Service Platform Release 3, 2003, http://www.osgi.org/
[49] M. Paolucci, T. Kawamura, T.R. Payne, K.P. Sycara. Semantic Matching of Web Services Capabilities. *In Int. Semantic Web Conference*, Sardinia, Italy, 2002.
[50] T.R. Payne, M. Paolucci, K. Sycara. Advertising and Matching DAML-S Service Descriptions. *In International Semantic Web Working Symposium*, Stanford University, California, USA, 2001.
[51] S. Rampacek. Sémantique, interactions et langages de description des services web complexes. Thèse de doctorat, Université de Reims Champagne-Ardenne, novembre 2006.
[52] P. Resnik. Using information content to evaluate semantic similarity in taxonomy. In *Proceedings of the international joint conference on artificial intelligence*, Canada, 1995.
[53] G. Salton. Developments in automatic text retrieval. 1991.
[54] A. Sajjanhar, J. Hou, Y. Zhang. Algorithm for web service matching. In *Proceedings of the 6th Asia-Pacific Web Conference,*2004.
[55] A. Sheth, C. Bertram et al. Managing Semantic Content for the Web, *IEEE Internet Computing*, 2002. http://lsdis.cs.uga.edu/library/download/S+2002-SCORE-IC.pdf
[56] K. Sivashanmugam, K. Verma, A. Sheth, J. Miller. Adding semantics to web services standards. In *proceedings of the 1st International Conference on web services (ICWS'03)*, Las Vegas, Nevada, 2003.
[57] J. F. Sowa. Knowledge Representation: Logical, Philosophical, and Computational Foundations, 2000.
[58] K. Sycara, S. Widoff. LARKS: Dynamic matchmaking among heterogeneous software agent in cyberspace. Autonomous Agents and Multi-Agent Systems. 2002.
[59] A. Tari. Conception d'une infrastructure robuste pour les services web. Thèse de doctorat, Université de Béjaia, Algérie, 2008.
[60] A. Tari, Islam ELDJADWI and Abdenasser DAHMANI: *A Dual-Layered Model for Web Services Representation and Composition***.** Journal of Intelligent Information System (JIIS**)**
[61] Z. Tari, N. Gooneratne Matching Independent Global Constraints for Composite Web Services. School of Computer Science and Information Technology RMIT University Melbourne, VIC 3001, Australia
[62] R. Troncy. Formalisation des connaissances documentaires et des connaissances conceptuelles à l'aide d'ontologies : application à la description de documents audiovisuels. Thèse de doctorat d'informatique de l'Université Joseph Fourrier – Grenoble I, 2004.

ANNEXE A. Les automates d'états finis

1. Introduction

Un automate fini (on dit parfois machine à états finis), en anglais *finite state automaton* ou *finite state machine* (FSA, FSM), est une machine abstraite utilisée en théorie de la calculabilité et dans l'étude des langages formels. C'est un outil fondamental en Informatique, où il intervient notamment en compilation des langages informatiques.

Un automate est constitué d'états et de transitions. Son comportement est dirigé par un mot fourni en entrée : l'automate passe d'état en état, suivant les transitions, à la lecture de chaque lettre de l'entrée. L'automate est dit « fini » car il possède un nombre fini d'états distincts.

Un automate fini forme un graphe orienté étiqueté, dont les états sont les sommets et les transitions les arêtes étiquetées.

2. Définitions formelles

2.1. Un automate d'états finis déterministe

Un automate d'états finis déterministe est un quintuplet $A=(\Sigma,Q,\delta,q_0,F)$ tel que :

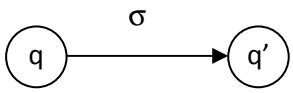
- Σ est un alphabet fini.
- Q est un ensemble fini d'états de A.
- $q_0 \in Q$ est l'état initial.
- $F \subseteq Q$ est l'ensemble des états finaux.
- $\delta : Q \times \Sigma \rightarrow Q$ est la fonction de transition de l'automate.

2.1.1. La représentation de l'automate déterministe

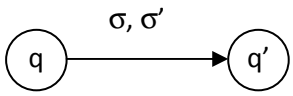
2.1.1.1. La représentation par diagramme

Un FSM A est représenté de la manière suivante :

Les états de A sont les sommets d'un graphe orienté et sont représentés par des cercles. Si $\delta(q,\sigma) = q'$ tel que $q, q' \in Q$ et $\sigma \in \Sigma$, alors un arc orientée de q vers q' et de label σ est tracé.



Les états finaux sont repérés grâce à un double cercle et l'état initial est désigné par une flèche entrante sans label. Enfin, si deux lettres σ et σ' sont telles que $\delta(q, \sigma) = q'$ et $\delta(q, \sigma') = q'$, alors un arc unique portant les deux labels séparés par une virgule est représenté.



Cette convention s'adapte aisément à plus de deux lettres.

L'automate prend un mot (constitué de symboles de son alphabet) en entrée et démarre dans son état initial. Il parcourt ensuite le mot en utilisant la fonction de transition δ afin de déterminer, pour chaque lettre, le prochain état à parcourir en utilisant l'état courant et le symbole venant d'être lu.

Si, à la fin de la lecture, la machine est dans un état accepteur, on dit qu'elle accepte l'entrée. Autrement, on dit qu'elle la rejette. L'ensemble des entrées acceptées forme un langage L, qui est le langage « reconnu » par l'automate A.

2.1.1.2. La représentation par une table

	σ_1	σ_2
q_0	$\delta(q, \sigma_1)$	…..
q_1	…..	…..

2.2. Automate fini non déterministe

Un automate fini non déterministe (AFN) A est un quintuplet $A = (\Sigma,,,_0,F)$ où :

- Q est un ensemble fini d'états,
- Σ est un alphabet,
- δ : Q × ($\Sigma \cup \{\varepsilon\}$) -> *P(Q)* est une fonction de transition,
- $I \subseteq Q$ est un ensemble non vide d'états initiaux,
- et $F \subseteq Q$ est un ensemble d'états terminaux.

ε est la séquence de taille nulle, aussi appelée « mot vide ».

2.3. Automate fini non déterministe généralisé

Un automate fini non déterministe généralisé (AFNG) A est un quintuplet $(\Sigma,,,_0,F)$ où :

- Q est un ensemble d'états,
- Σ est un alphabet,
- δ : (Q\{q})×(Q\{i}) ->R est une fonction de transition,
- q_0 est l'état initial,
- Et F est l'état accepteur.

R est l'ensemble de toutes les expressions rationnelles sur l'alphabet Σ.

3. Reconnaissance

Un mot $a_1a_2...a_n$ est reconnu par l'automate si et seulement si il existe une suite $k_0, k_1, ..., k_n$ d'éléments de Q tel que

$k_0 = q_0$

$k_n \in F$

$\forall i \in [1, n], \delta (k_i-1, a_i) = k_i$

4. Le langage accepté par un automate

Le langage reconnue par un tel automate est ;

$(A) = \{u \in \Sigma : \delta*(q_0,u) \cap F \neq \emptyset\}$

Ce qui se traduit par « il existe un chemin permettant d'atteindre $q_f \in F$ à partir de q_0 et u. »

5. L'équivalence de deux automates

Deux automates A et A' sont équivalents s'ils reconnaissent le même langage.

Théorème

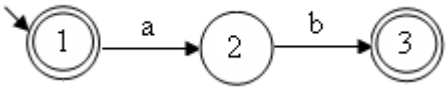
Tout automate fini non déterministe A est équivalent à un automate fini déterministe A' que l'on construit de façon canonique via un algorithme de déterminisme à partir de A.

6. Exemple de représentation d'un automate

Soit l'automate suivant : A=(Σ,Q,δ,q_0,F) où Q={1,2,3}, q_0= 1, F= {1,3}, Σ={a,b} et la fonction de transition δ est donnée par :

δ	A	b
1	2	∅
2	∅	3
3	∅	∅

L'automate A est représenté par la diagramme suivant



ANNEXE B. La réalisation de SWSMF

Pour la réalisation de l'approche SWSMF (Semantic Web Services Matching Framework), il y a plusieurs taches qui doivent être faites. Ces taches sont divisées en quatre grandes parties. Ces parties sont : les taches qui doivent être faites pour la création de l'ontologie utilisée ; celles qui concernent et doivent être faites par le service provider ; les taches accomplies par l'utilisateur ; enfin celles qui doivent être faites par le *matchmaker*. En rassemblant ces quatre parties ensemble nous donne la vue globale de la façon est manipulé le processus du matching par SWSMF.

La construction de l'ontologie :

SWSMF définit un méta-schéma pour la construction du domaine de l'ontologie, alors la personne responsable du développement des ontologies du domaine d'application doit adopter ce méta-schéma afin de construire des ontologies des domaines d'application valide. Il doit assurer :

- **l'identification du domaine d'application :** la personne qui s'en charge de la construction de l'ontologie doit enregistrer le domaine d'application dans le répertoire des domaines d'application afin de pouvoir récupérer son code.
- **l'identification des concepts du domaine d'application :** cette personne doit identifier les entités du domaine d'application, tel que les attributs.
- **l'identification du rôle du domaine d'application :** cette personne doit identifier les rôles du domaine d'application.
- **l'identification des opérations du domaine d'application :** elle doit identifier les transactions et leurs opérations correspondantes, et pour chaque opération, ses entrées, ses sorties, ses précontraintes et ses postcontraintes.
- **construire le graphe de substitution des concepts :** les fonctions de convertions, la matrice du mapping, doivent être définies pour chaque concept.

Le service provider

Le service provider doit accomplir les tâches suivantes :

- **l'identification du domaine d'application :** il doit identifier le domaine d'application de chaque service publié.
- **l'identification de l'ontologie utilisée :** il doit identifier l'ontologie de chaque domaine d'application.
- **l'identification des buts du service :** il doit identifier les opérations qui représentent le but pour chaque service.
- **l'identification des contextes fonctionnels de haut niveau :** les contextes fonctionnels doivent être définis pour chaque but, en définissant leurs précontraintes, postcontraintes et describing-contraintes correspondantes.
- **l'identification des rôles de l'utilisateur :** le rôle est identifié en définissant les secteurs visés par chaque service.
- **l'identification des GAP attendus :** le service provider doit définir les interactions des scénarios attendues entre le service et la requête de l'utilisateur.
- **l'identification le reste des spécifications :** il doit identifier les spécifications fonctionnelles de bas niveau, les spécifications non fonctionnelles et les spécifications non techniques pour chaque but.
- **construire SAV (Service Active View) et SPV (Service Passive View)**
- **la publication du service**

L'utilisateur

L'utilisateur doit former sa requête en respectant aux spécifications du SWSMF, il doit accomplir les tâches suivantes :

- **l'identification du domaine d'application et l'ontologie :** l'utilisateur doit sélectionner le domaine d'application et l'ontologie utilisée.
- **l'identification du but désiré :** il doit choisir les applications qui représentent le but désiré.
- **l'identification des contextes fonctionnels de haut niveau : l**es contextes fonctionnels de haut niveau doivent être définis pour chaque but par les précontraintes, postcontraintes et les describing-contraintes.
- **l'identification des rôles de l'utilisateur :** cela est fait pour chaque but défini.
- **l'identification le reste des spécifications.**
- **construire et soumettre UAV (User Active View) pour le *matchmaker*.**

L'utilisateur qui ne peut pas construire UAV, des buts supportés par l'ontologie choisie peuvent être proposés pour sélectionner un but.

Le *matchmaker*

Le *matchmaker* examine UAV construit avec le SAV construits. Pour que le *matchmaker* peut fonctionner, on doit avoir le code du domaine d'application, le répertoire des ontologies utilisées, un schéma du matching et enfin, au moins un SAV pour chaque service et un UAV pour la requête.