République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mentouri Constantine
Faculté des Sciences de l'Ingénieur
Département d'Informatique

N° ordre :
Série :

Thèse

Pour obtenir le diplôme de
Doctorat en sciences en Informatique

Titre

Composition des Web Services Sémantiques dans les systèmes Peer-to-Peer

Présentée par :

Mohamed GHARZOULI

Devant le jury :

Président ***: Pr. Benmohamed Mohamed****, Professeur à l'Université Mentouri, Constantine*

Rapporteur : ***Pr. Boufaida Mahmoud****, Professeur à l' Université Mentouri, Constantine*

Examinateur : ***Pr. Ahmed Nacer Mohamed****, Professeur à l'USTHB, Alger*

Examinateur : ***Dr. Belala Faiza****, Maitre de conférences, Université de Constantine*

Examinateur : ***Dr. Tari Abdelkamel*** *, Maitre de conférences, Université de Bejaïa*

Soutenue le : 25/09/2011

Remerciements

« Après avoir remercié ALLAH le tout puissant »

En premier lieu, je tiens à exprimer ma profonde reconnaissance au Professeur Mahmoud BOUFAIDA qui m'a formé à la recherche et a assuré avec beaucoup de gentillesse et d'attention l'encadrement de ma thèse de doctorat. J'ai pu à plusieurs reprises apprécier et profiter de sa haute compétence scientifique et pédagogique. Qu'il trouve dans ces lignes si courtes, la sincère expression de ma gratitude et le témoignage de ma reconnaissance.

Je tiens à remercier profondément le Professeur Mohamed Benmohamed qui m'apporter son soutien et me faire l'honneur de présider le jury de cette thèse.

C'est avec un grand plaisir que je compte le Professeur Mohamed AHMED NACER de l'USTHB parmi les membres de ce jury. Qu'il trouve l'expression de ma sincère reconnaissance pour être participer à ce jury.

Je suis également tout à fait honorée de la présence à ce jury du Mme Faiza Belala, pour m'avoir fait l'honneur participer à ce jury en qualité d'examinateur.

J'adresse mes sincères remerciements au docteur Tari Abdelkamel, Maitre de conférences à l'université Mira Abderahmene de Bejaïa, pour m'avoir fait l'honneur de participer à ce jury en qualité d'examinateur.

Merci à tous mes collègues du département d'Informatique pour leur soutien et leurs précieux encouragements,

Merci à tous mes collègues du laboratoire LIRE, ceux de l'équipe SIBC, particulièrement Ilhem Labed.

Merci à mes étudiants : Khalfallah Malik, Kamel Oussama, Kireche Med Seif Eddine, LABED Sami Amine et SOLTANE Amir Khalid.

Un grand merci à ceux qui ont une empreinte dans ce travail

Dédicace

A ma mère, à ma mère, à ma mère et à mon père

A mes sœurs

A mes amis et à mes collègues

A tous *ceux qui* me sont chers, …

Table des matières

Chapitre 3 : Une stratégie de découverte et de composition des Web services

Chapitre 4 : Une architecture distribuée pour la découverte et la composition des Web services

Table des illustrations

Figures

Tableaux

Introduction générale

1. Contexte

Avec une évolution exponentielle dans le temps, les SI (Systèmes d'Informations) sont devenus plus complexes et plus hétérogènes en raison de la diversité des besoins, des exigences des clients, et de la grande masse d'information stockées et manipulées par les internautes. Les infrastructures publiques et privées du secteur informatique sont de plus en plus multiplateformes, multifournisseurs et distribuées à grande échelle. Dans une telle situation, nombreux sont les professionnels de l'informatique qui considèrent que l'interopérabilité est un aspect aussi important que la sécurité et la fiabilité pour la gestion de leurs SI et leurs environnements de fonctionnement.

Assurer l'interopérabilité est l'une des clés de succès de développement et d'intégration des grands systèmes d'information. Cet objectif présente le noyau des travaux de recherches dédiés à l'amélioration des architectures, des plateformes et des technologies de développement des systèmes d'information depuis les méthodes cartésiennes jusqu'aux architectures orientées services, plus connue avec l'abréviation SOA[1] (pour Service Oriented Architecture). Ces dernières présentent actuellement la vision architecturelle des SI modernes.

SOA est un style architectural qui permet de construire des solutions d'entreprises basées sur les services. Historiquement, l'apparition du terme « service » est fortement liée à l'utilisation de l'architecture par composants comme approche d'intégration des applications **[PFI+96]**. Les composants sont des entités logicielles indépendantes fondées sur une interface et une sémantique bien définie. Ils interagissent moyennant une infrastructure qui permet de gérer la communication entre des composants au sein d'un même système ou à travers un réseau via une décomposition de la logique applicative en composants distribués **[Bro96]**, **[Szy97]**, **[MEL04]**.

Au début, la mise en place d'une architecture SOA a été basée sur les technologies et les middlewares conçus pour le modèle « objets distribués » comme CORBA, RMI et DCOM. Cependant, chacune de ces technologies proposent sa propre infrastructure, ce qui impose une forte liaison entre les services fournis et leurs consommateurs. Ainsi, on ne peut assembler entre eux que les composants de « même famille ». De ce fait, les systèmes résultants sont monolithiques **[MEL04]**, ce qui engendre le problème initial lié à l'intégration des systèmes hétérogènes.

En plus de l'urbanisation de son SI interne, l'entreprise d'aujourd'hui a besoin d'une ouverture vers son environnement économique pour accomplir des échanges interentreprises

1. Nous retiendrons l'acronyme SOA tout au long de ce document

avec ses partenaires par le biais d'Internet (B2B : Business to Business), et répondre le plus rapidement possible aux besoins de ses clients (B2C : Business to Consumer).

Actuellement, les solutions EAI (pour Enterprise Applications Integration) sont orientées vers les processus métiers et les échanges interentreprises. Elles prennent en compte les nouveaux modèles économiques créés et promus par Internet et ses technologies (TCP/IP, SMTP, HTTP, FTP, XML, etc.). De plus, les progiciels de gestion intégrés de la deuxième génération apportent une quantité de nouveaux modules organisés autour de la SOA **[RIV00], [BUR02]**. C'est dans ce contexte que les Web services sont apparus pour mettre en place cette architecture.

A la différence des technologies précédentes, les Web services proposent une architecture par composants qui permet à une application de faire l'usage d'une fonctionnalité située dans une autre application. Cependant, cette solution repose sur l'ubiquité de l'infrastructure d'Internet alors que les autres architectures reposent chacune sur sa propre infrastructure **[MEL04]**. L'interopérabilité est donc une caractéristique intrinsèque aux Web services parce qu'ils sont basés sur des technologies Web dérivées du fameux standard XML.

Depuis la naissance du paradigme des ***Web services*** en l'an 2000, ils sont été considérés comme une révolution pour le Web. En effet, avec les trois premières technologies de base : SOAP (Simple Object Access Protocol), UDDI (Universal Description Directory and Integration) et WSDL (Web services Description Language), les Web services ont connu un grand succès grâce au haut degré d'interopérabilité fourni par les standards. Les Web services sont présentés comme le meilleur moyen pour concrétiser la SOA (Service Oriented Architecture) et pour mettre en place le paradigme SOC (Services Oriented Computing) **[ORL+03]**, **[PAP+07]**. Cette technologie est utilisée actuellement dans plusieurs domaines comme les processus métiers et les applications scientifiques. Son objectif est de permettre une intégration dynamique des systèmes hétérogènes et de faciliter la coopération et la collaboration de plusieurs participants dans un environnement fortement distribué et hétérogène.

Cependant, les Web services possèdent aussi des inconvénients. En effet, il se peut qu'un client ait des besoins spécifiques qui ne sont rendus par aucun Web service. A cause de l'élargissement d'internet, il est rare pour un consommateur de découvrir un Web service qui réponde à ses besoins. Pour cette raison, plusieurs services peuvent interagir et échanger dynamiquement des informations. L'objectif de cette interaction est l'accomplissement d'un

but complexe non concevable par un seul Web service. Cette agrégation des compétences pour réaliser un but commun est connue par « **Composition des Web services** ».

Pour composer des Web services, ces derniers devraient être capables de découvrir d'autres services qui ont des capacités bien définies et qui réalisent des tâches précises. Pour garantir cela, il faut que l'infrastructure du système de recherche des Web services fournisse une description détaillée des fonctionnalités offertes par les services disponibles. Cette description doit être compréhensible par la machine pour faciliter la recherche dynamique et pour trouver les services adéquats. Ces défis, souvent rencontrés dans les travaux de recherche des Web services, sont fortement liés à l'opération de « **découverte des Web services »**.

Dans l'architecture des Web services, la découverte des Web services rencontre deux problèmes majeurs. Le premier est lié au point central de publication et de découverte implémenté généralement par un registre UDDI. Cette méthode centralisée est malheureusement mal adaptée aux interactions dynamiques dans un environnement évolutif (scalable) et flexible **[MAN+07], [HU+05]**. Aussi, une telle architecture réduit les performances du système global à cause du seul point d'échec (UDDI) et du coût élevé de la maintenance **[RAG08]**.

Le deuxième problème lié au mécanisme de découverte des Web services est lié à la description technique fournie par le contrat publié dans l'annuaire UDDI. Un Web service est décrit par une interface afin d'exposer ses fonctionnalités et ses capacités au monde extérieur (plus précisément aux consommateurs des services). Cette interface est généralement présentée par la description WSDL du service. Cette dernière fournit des données concernant le service à savoir : les opérations proposées, les paramètres d'entrées et de sorties, les messages, …etc.

Cependant, comme nous l'avons déjà précisé, les Web services sont conçus pour être composés afin d'accomplir des buts complexes non réalisables par un seul Web service. Pour aboutir à ces objectifs, les Web services doivent interagir dynamiquement avec le minimum d'intervention humaine. Ceci nécessite la prestation d'une description plus intelligible et plus compréhensible par la machine. C'est dans ce contexte que « **le Web sémantique** » peut intervenir pour régler le problème de découverte basé sur la description technique des Web services (réalisée en WSDL). En effet, une description sémantique est plus riche et plus compréhensible par la machine, ce qui facilite la découverte dynamique des Web services. La combinaison entre les Web services et les technologies du Web sémantique ont donné lieu à la naissance des « **Web services sémantiques** ».

Avec les avantages apportés par les technologies du Web sémantique à l'opération de découverte des Web services, UDDI ne saisit pas les relations entre les entités dans son répertoire et n'est donc pas capable de faire usage de l'information sémantique pour déduire les relations en cours de la recherche **[BAT+10]**. De plus, UDDI supporte uniquement la recherche basée sur des informations de haut niveau spécifiques aux entreprises et aux services. Il ne reçoit pas les spécificités des capacités des services pendant le mapping (Machmaking) **[SCH+04]**.

Afin de régler ce problème, plusieurs solutions ont été proposées pour procéder à la découverte distribuée des Web services. La plupart des travaux de recherche récents présentent des méthodes de découverte basées sur **les systèmes P2P (Peer-to-Peer) [MAN+07]**. Le paradigme P2P est considéré comme une évolution pour les systèmes distribués qui se concentrent sur la gestion des réseaux et le partage des ressources avec une meilleure fiabilité et scalabilité. Récemment, les systèmes P2P ont été impliqués dans plusieurs domaines de recherche comme le travail collaboratif et la découverte des ressources pertinentes y compris les Web services.

Les systèmes P2P fournissent une alternative évolutive aux systèmes centralisés en distribuant les Web services sur tous les pairs du réseau. Les approches fondées sur les systèmes P2P offrent un contexte décentralisé et auto-organisé, où l'interaction entre les Web services se fait dynamiquement.

2. Problématique

L'utilisation et la combinaison des technologies pour résoudre des problèmes dans un contexte bien défini n'est pas un travail facile ou toujours réalisable. Récemment, plusieurs solutions ont été proposées pour procéder à la découverte distribuée des Web services. Les solutions proposées sont cependant fortement liées au type de réseau P2P utilisé : centralisé, hybride, structuré ou non structuré. Chaque type de réseau utilise ses propres protocoles de communications et ses méthodes de recherche les plus appropriées.

De ce fait, pour employer les systèmes P2P dans le domaine de la découverte et composition des Web services sémantiques, il faut répondre à plusieurs questions:

Selon les avantages et les inconvénients de chaque famille des systèmes P2P, quel est le type de réseau P2P le plus adapté à la découverte dynamique des Web services ?

Plusieurs travaux de recherches proposent la découverte et la composition des Web services dans les réseaux P2P structurés comme Chord **[STO+01]**, Pastry **[ROW+01]** et CAN

[RAT+01]. Ce type de réseau a apporté plusieurs solutions aux problèmes rencontrés dans les premières générations des réseaux P2P (centralisés et non structurés). Cependant, les réseaux structurés sont basés généralement sur les protocoles qui emploient une table de hachage DHT (Data Hash Table). Cette dernière ne peut pas permettre des recherches complexes ou des recherches par contenu. D'autre part, les protocoles basés DHT sont difficiles à mettre en place à cause de la typologie statique imposée (par exemple anneau dans Chord).

Par ailleurs, il existe très peu de travaux de recherches qui utilisent les réseaux non structurés pour la découverte des Web services. Alors que, les systèmes non structurés (comme Gnutella V0.4 **[GNUv04]**) ne nécessitent ni des supers nœuds centralisés ni un contrôle précis sur la topologie du réseau ou le placement des données. En plus, ils sont très adaptés à la recherche par contenu. Bien que l'algorithme de recherche (basé sur les inondations), utilisé dans ces systèmes, pose un problème de passage à l'échelle **[LV+02]**.

Comment peut on assurer la compatibilité et l'interopérabilité technique et sémantique des Web services distribués dans un système fortement distribué et hétérogène comme les systèmes P2P ?

Dans ce contexte, les Web services sont implémentés et distribués sur les différents nœuds du réseau où chaque fournisseur peut utiliser ses propres concepts et son propre langage de description sémantique. Particulièrement, nous signalons que dans les systèmes P2P les ressources et les Web services sont fournis par des simples internautes plutôt que par des entreprises.

Comment découvrir des Web services décris sémantiquement pour répondre à une requête précise tout en assurant une composition dynamique ?

Autrement dit, comment implémenter une méthode de découverte distribuée ? et comment peut on exploiter les technologies du Web sémantique pour minimiser le plus possible l'intervention humaine ? Dans ce contexte, il faut que les requêtes échangées, entre les différents nœuds du réseau, soient compréhensibles par tous les participants.

Comment minimiser le temps de réponse et le coût de composition ?

Suivant le type de réseau, sa taille et les protocoles utilisés, le temps de réponse peut différer d'une solution à une autre. Cependant, l'objectif principal de toute méthode de découverte des Web services est de minimiser le temps de recherche et de réponse. D'autre part, dans le contexte des systèmes P2P, le passage à l'échelle est un facteur important qu'il faut prendre en considération parce qu'il influe directement sur le coût de l'opération de recherche et par voie de conséquence sur celui de la composition.

3. Objectifs et contributions de la thèse

Dans notre travail, nous nous intéressons aux systèmes décentralisés, spécialement aux réseaux non structurés. Ces derniers présentent plusieurs avantages, en plus à son utilisation par des très grandes communautés d'internautes, les réseaux P2P non structurés possèdent des caractéristiques qui les positionnent comme étant le type de réseau le plus approprié à la découverte des Web services sémantiques. Généralement, dans ce contexte, le demandeur (le service consommateur) recherche une capacité, un but ou une propriété d'une ressource (Web service). Ainsi, quand il envoie une requête, le demandeur n'a pas besoin de connaître préalablement l'emplacement ou l'identificateur de la ressource. Cette caractéristique n'est pas fournie par les systèmes P2P structurés. Pour retrouver une ressource, il est nécessaire que le demandeur connaisse à l'avance l'identifiant de la ressource.

Nous rappelons que l'objectif principal de notre travail est de fournir un scénario purement décentralisé qui supporte la composition automatique des Web services sémantiques distribués sur un réseau P2P. Notre contribution afin d'atteindre cet objectif s'articule autour des points suivants :

La première étape de la solution consiste à proposer un algorithme distribué pour la découverte et la composition des Web services sémantiques. L'objectif est de créer un espace collaboratif entre les différents pairs participants à la réalisation d'un but commun.

Dans cette étape, nous avons décrit une stratégie de découverte et de composition des Web services dans un réseau P2P non structuré. L'idée principale est de présenter un algorithme épidémique permettant la rechercher des services distribués sur tous les pairs du réseau afin de composer de nouveaux services personnalisés.

Pour cela, nous avons utilisé une table dite table de composition afin de préserver la trace du chemin de composition pour une éventuelle future réutilisation. En effet, la réutilisation des compositions déjà réalisées présente un facteur important qui peut minimiser le temps et le coût de réponse. En raison de la nature volatile des pairs dans un réseau P2P, nous avons mis en place une méthode permettant de maintenir la cohérence des données de la table de composition.

La deuxième partie de notre travail consiste à proposer une architecture qui implémente la stratégie présentée ci dessus. Pour bien le faire, nous avons proposé un modèle qui combine les avantages des méthodes centralisées et décentralisées. Nous avons décrit un framework totalement distribué pour implémenter l'algorithme de découverte, alors que nous avons

préféré centraliser un référentiel qui contient une ontologie globale afin d'assurer l'interopérabilité sémantique. Pour accomplir cette tâche nous avons employé les langages de descriptions ontologiques OWL et OWL-S. Nous avons aussi enrichi cette architecture par un processus améliorant le fonctionnement de cette dernière.

La dernière étape de notre travail consiste à concrétiser la solution proposée à travers une implémentation. Pour atteindre ce but, nous avons eu recours à plusieurs outils issus des technologies du Web sémantique et celles des systèmes P2P. Une étude de cas relative a été présentée pour motiver la solution proposée dans les étapes précédentes.

4. Organisation de la thèse

Le manuscrit est structuré en cinq chapitres et une conclusion générale.

Le premier chapitre est consacré à la présentation des systèmes P2P et leurs applications. Nous montrons les différents types des réseaux P2P, leurs modes de fonctionnement, les protocoles utilisés par chaque type ainsi que les avantages et les inconvénients de chaque système.

Dans le deuxième chapitre, nous présentons les Web services et les technologies adjacentes. Après avoir étudié le mécanisme de couplage entre le Web sémantique et les Web services, nous décrivons les différents langages de descriptions sémantiques dédiés aux Web services.

Dans le chapitre lié à notre contribution, nous présentons une stratégie générique pour la découverte distribuée des Web services sémantiques. Dans cette stratégie nous employons les systèmes P2P non structurés pour proposer un algorithme épidémique de recherche. Nous proposons une table distribuée dite « table de composition » pour préserver les traces des compositions déjà réalisées dans le réseau. Nous montrons les avantages de cette table tout en assurant la cohérence de son contenu à travers des algorithmes de notification.

Le chapitre 4 illustre l'architecture PM4SWS qui implémente la stratégie présentée dans le chapitre 3. Nous détaillons les différents composants de cette architecture et les interactions entre eux en faisant référence à l'algorithme de découverte présenté dans le chapitre précédent. Un processus de développement a été proposé pour améliorer le fonctionnement de cette architecture.

Le chapitre 5 illustre l'application de notre approche à une étude de cas. Nous décrivons tout d'abord les différents outils techniques utilisés. Nous précisons ensuite l'implémentation

des différents composants inclus dans notre architecture et comment les opérations de découverte et de composition peuvent être réalisées.

Enfin, le manuscrit se termine par une conclusion générale qui récapitule les travaux réalisés et propose quelques visions pour les travaux futurs.

Chapitre 1

Systèmes P2P

1. Introduction

Historiquement, plusieurs facteurs ont contribué à l'apparition des systèmes distribués. Sur le plan matériel, les ordinateurs sont de plus en plus petits et de moins en moins chers, grâce à la commercialisation rapide et compétitive des machines personnelles (PC). D'autre part, le développement des logiciels a connu une évolution formidable avec l'apparition des systèmes d'exploitation conçus pour les petites machines, ce qui a donné lieu à la construction de réseaux de communication reliant différentes entités jusqu'à la naissance du réseau mondial connu sous le nom d'Internet.

Par ailleurs, Internet a connu son succès avec la plus grande application dans ce réseau qui est le World Wide Web (WWW). Cette dernière est le fruit de trois technologies de base : HTML, le protocole HTTP et le modèle Client/serveur. Ce dernier représente l'architecture de base sur laquelle ont été conçu la plupart des applications distribuées connues aujourd'hui.

Cependant, l'évolution rapide du rôle actif de l'Internet dans l'économie et la société, a exigé l'accessibilité à un maximum de ses ressources. Néanmoins, ces dernières, sont généralement mal exploitées, notamment les trois facteurs importants pour le fonctionnement des applications à grande échelle: bande passante, espace de stockage et capacité de traitement. Dans ce contexte, le modèle client/serveur classique a rapidement posé plusieurs limites : tout est localisé sur la même machine et accessible par le programme, le système logiciel s'exécutant sur une seule machine ; et accédant localement aux ressources nécessaires (données, code, périphériques, mémoire ...). C'est parce que l'architecture client/serveur ne parvient pas à tirer le maximum des ressources du réseau, que le modèle Pair à Pair (Peer to Peer ou P2P) est né.

Dans ce chapitre, nous allons décrire les différents concepts des réseaux P2P. Nous y fournissons quelques définitions, les motivations d'utilisation des ces systèmes et leurs principales caractéristiques. Ensuite, nous allons entamer la partie relative aux types d'applications des réseaux P2P. Enfin, nous détaillerons les différentes architectures des systèmes P2P, ainsi que les avantages et les inconvénients de chaque type d'architectures.

2. Concepts de base des systèmes P2P

2.1 Historique et Définitions

Historiquement, le P2P est devenu très populaire en 1999 aux Etats Unis, avec le logiciel de partage de musique en ligne Napster [**Napster]**. Rapidement, ce dernier a connu un grand

problème de copyright pour les compagnies et les éditeurs de disque, ce qui a mené Napster à des poursuites judiciaires en 2000.

Ce problème a été la cause principale de l'apparition des systèmes P2P totalement décentralisés, pour rendre le contrôle judiciaire plus difficile. Cependant, il y a des nombreuses utilisations légales qui sont aujourd'hui impliquées dans le monde du P2P, mais il est dur d'interdire l'échange de fichiers illégaux (musique, films...).

Comme mentionné ci-dessous, nous constatons que ce bref historique relève les grandes classes des systèmes P2P qui donnent lieu à la multitude de définitions liées au concept P2P que l'on retrouve dans la littérature. La plus stricte est celle donnée au modèle dit **pur** P2P : « *un système totalement distribué dans lequel tous les nœuds sont équivalents en termes de fonctionnalités et de tâches à résoudre*» **[AND+04].** Cette définition ne concerne pas d'autres modèles P2P comme KAZAA **[Kazza]** qui est largement connu comme un système P2P dit **hybride**.

Une deuxième définition plus large a été donnée par ***Shirky*** en 2000: « *le P2P est une classe d'applications qui exploite des ressources de stockage, de traitement, à travers le réseau Internet avec une présence humaine disponibles*» **[SHI00]**. Cette définition est plus générale car elle englobe les systèmes P2P qui se basent sur des serveurs centralisés comme SETI@home **[SETI]** et Napster **[Napster].**

Une autre définition plus significative des systèmes P2P est la suivante : « *Les systèmes pair à pair (P2P, de l'anglais "peer-to-peer") sont composés d'un ensemble d'entités partageant un ensemble de ressources, et jouant à la fois le rôle de serveur et de client. Chaque nœud peut ainsi télécharger des ressources à partir d'un autre nœud, tout en fournissant des ressources à un troisième nœud* » **[BEN06]**. Cette définition exprime les objectifs principaux des systèmes P2P qui sont le partage des ressources et le travail collaboratif.

2.2 Caractéristiques des systèmes P2P

Comme nous allons le présenter plus tard dans ce chapitre, il existe plusieurs types des réseaux P2P avec plusieurs architectures, protocoles et mode de fonctionnement. Cependant, quelques caractéristiques sont communes à tous ces systèmes, les plus importantes sont :

- Un système P2P doit être constitué d'au moins de deux pairs.
- Les nœuds d'un réseau P2P sont généralement de nature volatiles (les pairs peuvent joindre ou quitter le réseau en toute liberté).

- Un point ou plusieurs nœuds centralisés peuvent jouer le rôle des serveurs dédiés dans un système P2P, selon la nature de l'application **[LOO06].** Ces nœuds sont connus sous le nom de « Super-Pairs ». Les systèmes P2P qui ne contiennent pas de serveurs dédiés sont dits systèmes P2P purs.
- Les pairs peuvent appartenir à différents propriétaires. Dans le cas de « clusters », un pair peut être membre de plusieurs groupes en même temps.

2.3 Objectifs des systèmes P2P

Comme tout système informatique, les systèmes P2P ont pour principal objectif de répondre aux exigences des utilisateurs. Les réseaux P2P sont apparus pour résoudre quelques problèmes rencontrés dans le paradigme client/serveur. Les atouts pour lesquels les internautes préfèrent l'utilisation des systèmes P2P sont souvent les critères suivants **[MIL+03], [BEN06]**:

Partage et réduction des coûts entre les différents pairs: dans le paradigme client/serveur la majorité de l'effort et du budget est réservé au côté serveur.

Agrégation des ressources : elle permet l'amélioration du rendement des ressources en temps d'utilisation**.** Chaque nœud dans un système P2P partage des ressources comme la puissance de traitement et l'espace de stockage.

Anonymat: dans un système P2P, un utilisateur peut cacher quelconques informations concernant les autres.

Décentralisation : les nœuds ont tous le même niveau. Dans une telle situation, l'utilisateur possède et garde le contrôle total de ses ressources.

Autonomie: un utilisateur P2P peut éviter la connexion à aucun serveur centralisé dédié. Il peut aussi décider du degré de la transparence des données. Ses travaux peuvent être relatifs à un niveau local. Chaque pair a alors la responsabilité de partager ses ressources ou non.

Dynamisme : dans un système P2P, les nœuds peuvent joindre ou quitter le système librement, ce qui rend l'environnement plus flexible, évolutif et dynamique. Ce principe permet d'éviter les goulots d'étranglement et assure le passage à l'échelle du système.

Fiabilité et tolérance aux pannes : l'absence d'un élément central permet aux systèmes P2P de bénéficier d'une bonne tolérance aux pannes. Cette caractéristique est plus puissante dans les systèmes P2P purs (sans serveurs dédiés).

3. Avantages et inconvénients des systèmes P2P

Nous pouvons résumer les différents avantages et inconvénients des systèmes P2P comme suit :

3.1 Avantages

A partir de propriétés citées ci-dessus, nous pouvons affirmer que les systèmes P2P sont plus adaptés aux environnements qui possèdent de grandes quantités de ressources à partager. Par apport à une approche client/serveur, les systèmes P2P permettent de **[ANT+02]**:

- ✓ Eviter la création de goulots d'étranglement.
- ✓ Passer à l'échelle supérieure.
- ✓ Exploiter des ressources importantes distribuées sur un grand nombre de nœuds du réseau.
- ✓ Améliorer l'utilisation de la bande passante, des ressources de traitements et des espaces de stockage.
- ✓ Accélérer l'accomplissement des tâches en réduisant le temps de traitement à travers les liens directs entre les pairs.
- ✓ Eviter le seul point d'échec grâce à la distribution redondante des ressources et la décentralisation des systèmes P2P. Cette caractéristique offre à ce type de systèmes plus de fiabilité et de robustesse.
- ✓ Augmenter les performances du système en partageant la charge du travail et l'accroissement de l'autonomie et l'agrégation des ressources, ce qui augmente la performance des réseaux P2P.
- ✓ Permet aux utilisateurs de maintenir le contrôle de leurs ressources. En plus, ils peuvent joindre ou quitter le système à tout moment.

3.2 Inconvénients

Les réseaux P2P, présentent aussi des inconvénients majeurs à savoir :

- ✓ **Problème de sécurité** : la sécurité n'est pas toujours assurée lors des communications entre les nœuds du réseau à cause des pairs malveillants.
- ✓ **Pairs égoïstes** : des pairs qui exploitent les ressources des autres et n'offrent rien.

- ✓ **Difficulté d'administration** : la décentralisation rend le système difficile à administrer, et une connaissance globale de l'état des ressources et du réseau est presque impossible.
- ✓ **Hétérogénéité et faible interopérabilité**: Vue la diversité des architectures, des protocoles utilisés, en plus de l'absence de standardisation d'un système P2P à un autre, l'intégration et la communication entre les logiciels P2P n'est pas une tâche toujours faisable.
- ✓ **Problème de disponibilité des ressources** : lorsqu'un nœud quitte le réseau, toutes les ressources qu'il fournit disparaissent aussi.
- ✓ **Problème de publication et de découverte des ressources** : dans certains types de systèmes P2P, les mécanismes efficaces de publication et de découverte des ressources manquent particulièrement dans le cas où de nouveaux pairs joignent le réseau.

4. Types d'applications P2P

Depuis leur apparition, les réseaux P2P étaient considérés comme un système d'échange de fichiers. Jusqu'au maintenant, cette application présente le plus grand taux de trafic des différents réseaux P2P. C'est pour cette raison que le terme P2P peut parfois se confondre avec les systèmes d'échange de fichiers alors que ces derniers ne représentent qu'un cas de figure des types d'applications basés sur le paradigme P2P.

Les systèmes P2P sont utilisés dans plusieurs catégories d'applications qui peuvent être réparties en cinq classes **[ALE+06]**: calcul distribué, diffusion des données, travail collaboratif, moteurs de recherche et plateformes d'urbanisation.

4.1 Calcul distribué

Même si les ressources d'un ordinateur à part sont limitées, le regroupement de nombreux ordinateurs permet d'obtenir des performances théoriquement considérables. Ceci est l'un des objectifs des systèmes P2P largement illustré dans la littérature par l'exemple des mathématiciens (et physiciens) avec leurs calculs matriciels. Le but est de diviser un gros calcul, par exemple celui de l'inversion de la matrice en petits traitements plus ou moins indépendants que l'on pourrait répartir entre les pairs. Ce type d'applications est connu sous le nom de « GRID Computing ».

4.2 Diffusion de contenu

Le but de ce type d'applications est généralement le partage des fichiers qui présente l'objectif principal pour lequel les systèmes P2P ont vu le jour. Ces systèmes sont utilisés pour créer un réseau de nœuds permettant la recherche et le transfert des fichiers. Dans cette catégorie on retrouve des systèmes comme : Napster **[Napster]**, Gnutella **[gnutella]**, Freenet **[CLA+02],** KAZAA **[Kazza]**, ...

Le partage des fichiers dans un réseau P2P est souvent implémenté dans le cas ou des fichiers doivent être diffusés auprès d'un grand nombre de personnes (par exemple diffusion d'une émission vidéo ou audio), l'application P2P la plus adaptée à ce type d'application est BitTorrent **[BitTorrent]**.

Une deuxième fonctionnalité qui permet la diffusion de contenu est celle de stockage et de publication de contenu. Le but de ces systèmes est de créer un espace distribué qui permettra le stockage et la publication des données. Ces dernières seront utilisées par les pairs qui ont le privilège d'accès. Les exemples d'OceanStore **[KUB+00]** et Ivy **[MUT+02]** se distinguent dans cette catégorie.

4.3 Travail collaboratif

Ce type d'applications permet à un groupe d'utilisateurs de collaborer en temps réel sans utilisation d'un serveur central. Un exemple d'applications populaires qui utilisent des messages instantanés est Yahoo, AOL et Jabber **[ALE+06]**.

Dérivés des systèmes « groupware » utilisés au niveau des entreprises, ce type d'applications permet à un ensemble d'utilisateurs de travailler de manière commune sur un projet distribué. Les exemples les plus proches à cette catégorie sont : Groove **[Groove]**, Magi **[Magi]**, PowerPoint distribué **[PPD]** et NextPage **[NextPage]**, **[ALE+06]**.

Par exemple, Groove permet aux employés d'une même société qui travaillent sur un même projet, de partager des documents, des emplois du temps ou de communiquer en temps réel dans une architecture entièrement décentralisée et cryptée pour assurer la confidentialité de l'ensemble. Il peut être aussi utilisé par des partenaires commerciaux pour des échanges B2B (Busines to Business).

4.4 Moteurs de recherches

Le but est d'éviter les limites des moteurs de recherche centralisés qui n'explorent qu'un petit morceau du contenu de Web. En effet, le principal problème auquel sont confrontés les

moteurs de recherche vient du fait que de plus en plus de pages Web sont générées dynamiquement et que les informations qu'elles contiennent proviennent de bases de données auxquels les moteurs n'ont pas accès **[ALE+06]**.

Les moteurs de recherche P2P reposent sur une architecture complètement distribuée. On peut citer InfraSearch **[INFS]** comme moteur de recherches P2P. Ce type de moteur n'effectue pas lui-même la recherche et la récupération de l'information. Il ne fait que propager les requêtes formulées par l'utilisateur aux systèmes qui lui sont connectés, eux-mêmes les répercutant à leurs voisins sur le réseau. Chaque hôte recevant la requête a la responsabilité totale d'effectuer lui-même la recherche sur son propre système et de renvoyer ensuite à InfraSearch les informations jugées pertinentes **[ALE+06]**.

4.5 Plateformes

Les plateformes proposent une infrastructure générique pour développer des applications P2P en offrant les fonctions de base telles que la gestion de pairs, l'attribution d'identifiants, la découverte des ressources, la communication entre les pairs, la sécurité et d'autres fonctionnalités. Ainsi Sun propose sa plateforme **JXTA [JXTA]** basée sur la technologie Java, et Microsoft intègre dans **.NET [.NET]** des outils pour développer des applications P2P.

5. Différentes architectures P2P

Suivant les différents niveaux de décentralisation, nous pouvons classer les architectures P2P en trois catégories principales : centralisée, totalement décentralisée et hybride.

5.1 Architecture centralisée

Ce type d'architecture présente la première génération des réseaux P2P. Il repose sur un serveur central qui contient des métadonnées décrivant les ressources (spécialement des fichiers) partagées par les participants du réseau. Les utilisateurs se connectent au serveur afin de communiquer entre eux et d'échanger des fichiers.

Le rôle principal du serveur est la gestion de la recherche des ressources en identifiant les nœuds stockant les fichiers. Après la connexion au serveur et la découverte du nœud qui possède le fichier voulu, la communication et l'échange de ce fichier, entre le pair demandeur et le pair fournisseur, se fait de façon directe. La figure suivante présente cette architecture **:**

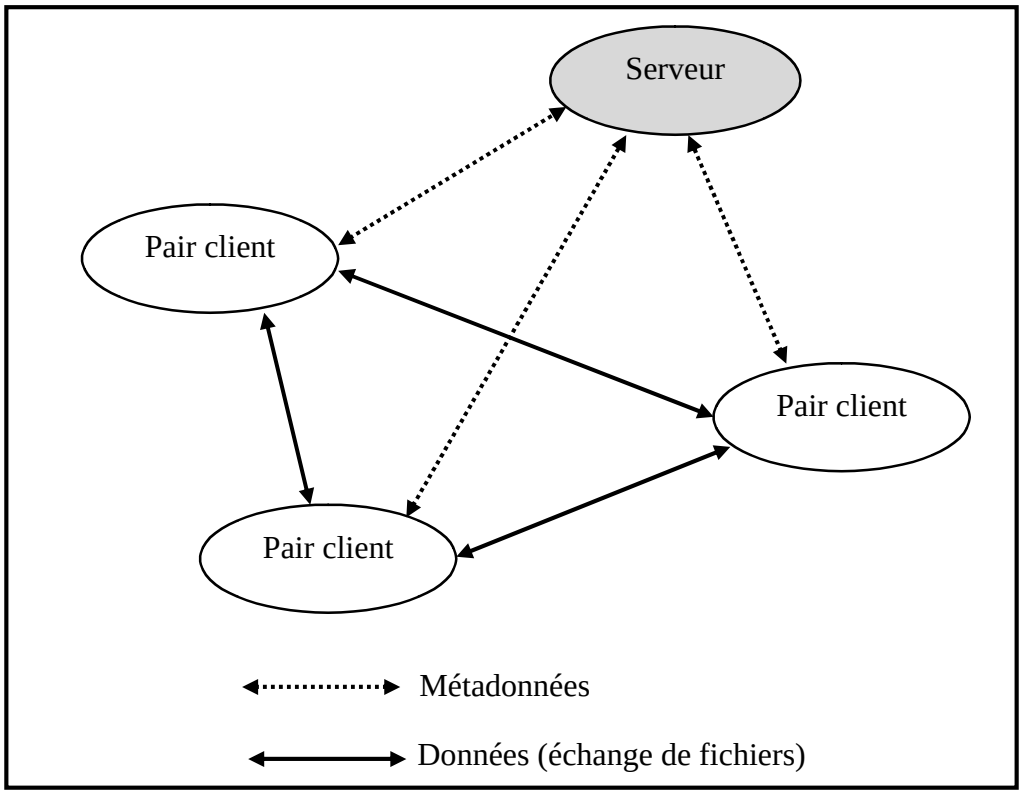


Figure1.1 : Architecture centralisée.

Dans cette architecture (exemple de NAPSTER), chaque pair héberge un ensemble de fichiers qu'il partage avec les autres pairs du réseau. Alors que le serveur sauvegarde deux types de données **[AND+04]** :

- Une table qui dispose d'informations sur les pairs connectés (adresse IP, numéro du port, bande passante disponible …).
- Une table qui recense tous les fichiers partagés par chaque pair, avec des métadonnées qui décrivent chaque fichier (nom du fichier, date de création, …).

Les étapes suivies par un utilisateur sont les suivantes :

- L'utilisateur se connecte au serveur central et annonce les fichiers qu'il partage,
- S'il cherche un fichier, il envoie une requête au serveur,
- Le serveur effectue une recherche dans sa table d'index et retourne la liste des pairs qui disposent du fichier recherché,
- Enfin, le pair client ouvre une connexion directe avec un ou plusieurs pairs hébergeant le fichier recherché et le télécharge directement sans transiter par le serveur.

Le logiciel P2P qui utilise cette architecture est NAPSTER. C'est le premier programme P2P conçu pour le partage de fichiers, spécialement pour l'échange de fichiers audio/vidéo**.**

5.2 Architecture décentralisée

Cette architecture est d'une nature plate où tous les nœuds du réseau remplissant les mêmes tâches. Ils jouent le rôle de serveur et de clients en même temps. Les pairs d'un tel réseau sont souvent appelés SERVENT (**SERV**eur+cli**ENT**). Cette architecture ne disposant pas d'un serveur central, du fait qu'un pair quitte le réseau n'affecte pas le système.

Nous pouvons désigner deux types d'architectures décentralisées : non structurés et structurés.

5.2.1. Architecture non-structurée

Historiquement, ce type d'architecture représente la deuxième génération des réseaux P2P. Une architecture non structurée ne possède pas une typologie spécifique. Chaque nœud a une vision limitée du réseau, il n'est connecté qu'à 3 ou 4 nœuds au plus, et connait tous les pairs qui se trouvent à une profondeur d'arbre généralement inférieure à 7. Lorsqu'un nouvel utilisateur veut joindre le réseau : il envoie un message ***broadcast*** pour savoir quels sont les autres nœuds du réseau qui sont connectés **[ALE+06]**.

La recherche des ressources se fait par une transmission d'une requête aux pairs voisins. Ces derniers la passe à leurs voisins et ainsi de suite (inondation du réseau). Afin de limiter la génération d'un nombre exponentiel de messages, les requêtes sont dotées d'une durée de vie (TTL : Time To Live) qui est décrémentée au niveau de chaque nœud par lequel transite la requête. De plus, les paquets transmis sont détectés grâce à leurs identificateurs, ce qui empêche la retransmission d'un seul paquet dans plusieurs chemins différents. Enfin, une fois la ressource trouvée, une connexion directe s'établit entre le pair demandeur et le pair fournisseur. La figure 1.2 présente l'architecture purement décentralisée.

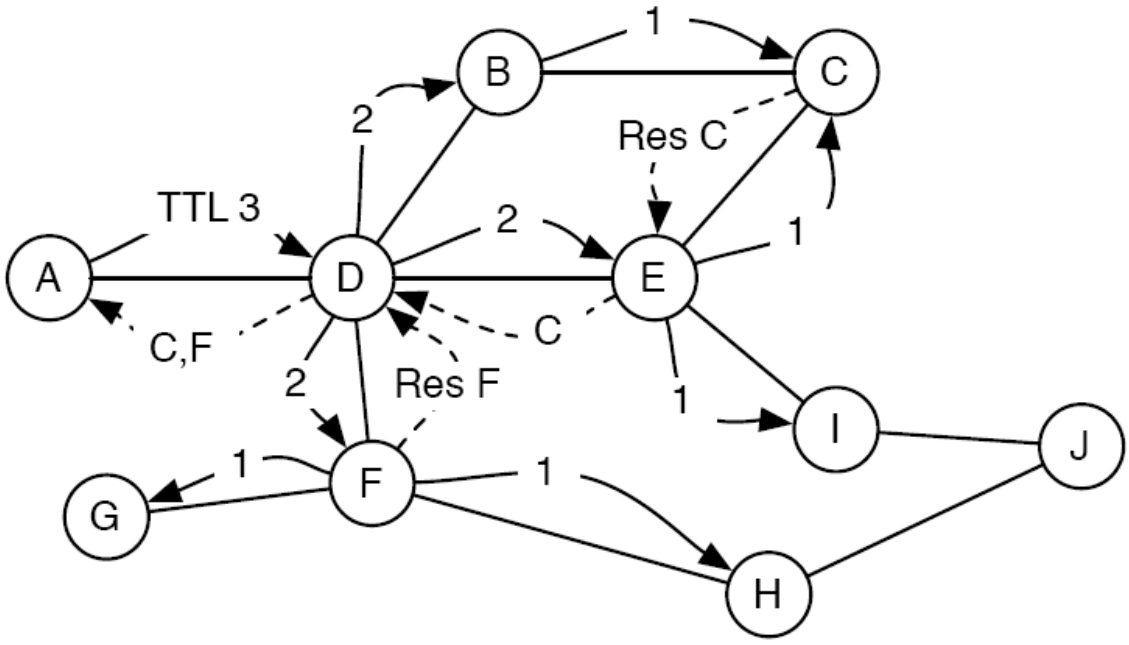


Figure 1.2 : Architecture non structurée (Exemple de Gnutella) [BEN06].

L'exemple typique de ce type d'architecture est celle de Gnutella v0.4 (figure 1.2) **[GNUv04]**. C'est un protocole décentralisé développé en mars 2000 par Tom Pepper et Justin Fränkel et initialement conçu pour le partage de fichiers.

Pour rejoindre le réseau, un pair doit connaitre au moins un nœud déjà présent sur le réseau (un servent), ensuite, il peut utiliser un **Ping** pour trouver les adresses d'autres servents. Chaque servent maintient une connexion avec environ 5 autres servents et un réseau logique se forme alors sur le réseau global. Un servent qui reçoit un Ping doit répondre avec un (ou plusieurs) **Pong**, indiquant son adresse IP et son numéro de port, ainsi que le nombre et la taille des fichiers partagés **[BEN06]**.

Si un nœud veut chercher une ressource, il envoie une requête à ses voisins. Ces derniers retransmettent le message (inondation). Une fois la ressource est trouvée, son adresse est propagée le long du chemin inverse. Enfin une connexion directe est établie entre le pair demandeur et le pair qui possède la ressource pour échanger cette dernière. Si les données sont derrière un firewall, un message **Push** est alors utilisé.

5.2.2. Architecture Structurée

Pour résoudre les problèmes qui sont posés dans les réseaux non structurés, un autre type d'architectures, purement décentralisées, est apparu. Il s'agit des systèmes P2P structurés qui utilisent une table distribuée dite **DHT « Distribued Hashed Table ».** Cette table permet de

fournir un routage efficace d'un point de système à un autre. Les nœuds possèdent seulement des connaissances locales, donc pas de connaissances globales, mais ils peuvent se rapprocher de la donnée recherchée.

Chaque ressource dans le réseau est repérée par une clé. Les DHT, distribuées sur l'ensemble des pairs, conservent les valeurs correspondant à leur domaine. A l'insertion, le couple (clé, valeur) est placé sur le pair responsable de la clé et non sur le pair l'ayant inséré. Pour récupérer une clé, un nœud doit interroger le pair qui en est responsable. Ce dernier lui fournit le nœud à interroger pour obtenir la donnée associée à la clé.

Sous cette catégorie, nous pouvons trouver plusieurs exemples de systèmes structurés comme CAN **[RAT+01]** et PASTRY **[ROW+01]**. Cependant l'exemple le plus cité dans les travaux de recherche est le protocole CHORD **[STO+01]**. Dans ce dernier, chaque ressource K est associée au nœud N lui correspondant, comme illustré dans la figure suivante :

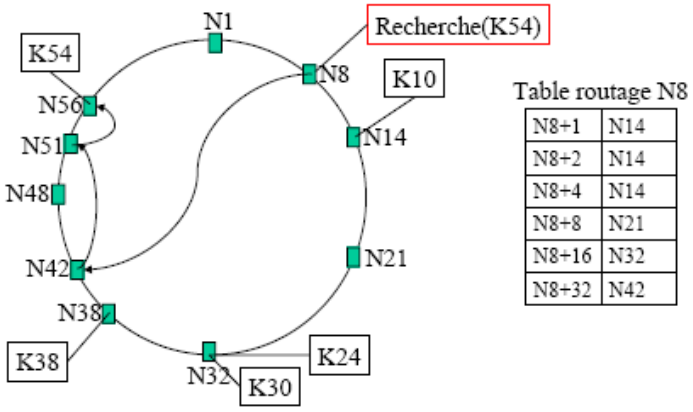


Figure 1.3: **Recherche dans CHORD [STO+01]**.

- La donnée ayant la clé 54 est recherchée sur le nœud N8.
- N8 consulte sa table de routage et voit que le nœud le plus proche de 54 est 42.
- N8 transmet la requête à N42 qui recommence le même processus.

5.3 Architecture partiellement centralisée : « Hybride »

Pratiquement, les nœuds d'un réseau P2P ont des capacités de stockage et de traitement caractérisées par une forte disparité ce qui approuve la création du modèle « hybride » qui est un couplage entre le mode P2P et l'architecture Client/serveur.

Les réseaux hybrides utilisent un nombre suffisant de serveurs (dits super-pairs) pour éviter le risque en cas de disparition de l'un d'eux. Chaque serveur présente un nœud central d'un groupe (cluster) qui comporte un ensemble de pairs organisés en P2P.

La figure suivante présente clairement l'architecture partiellement centralisée :

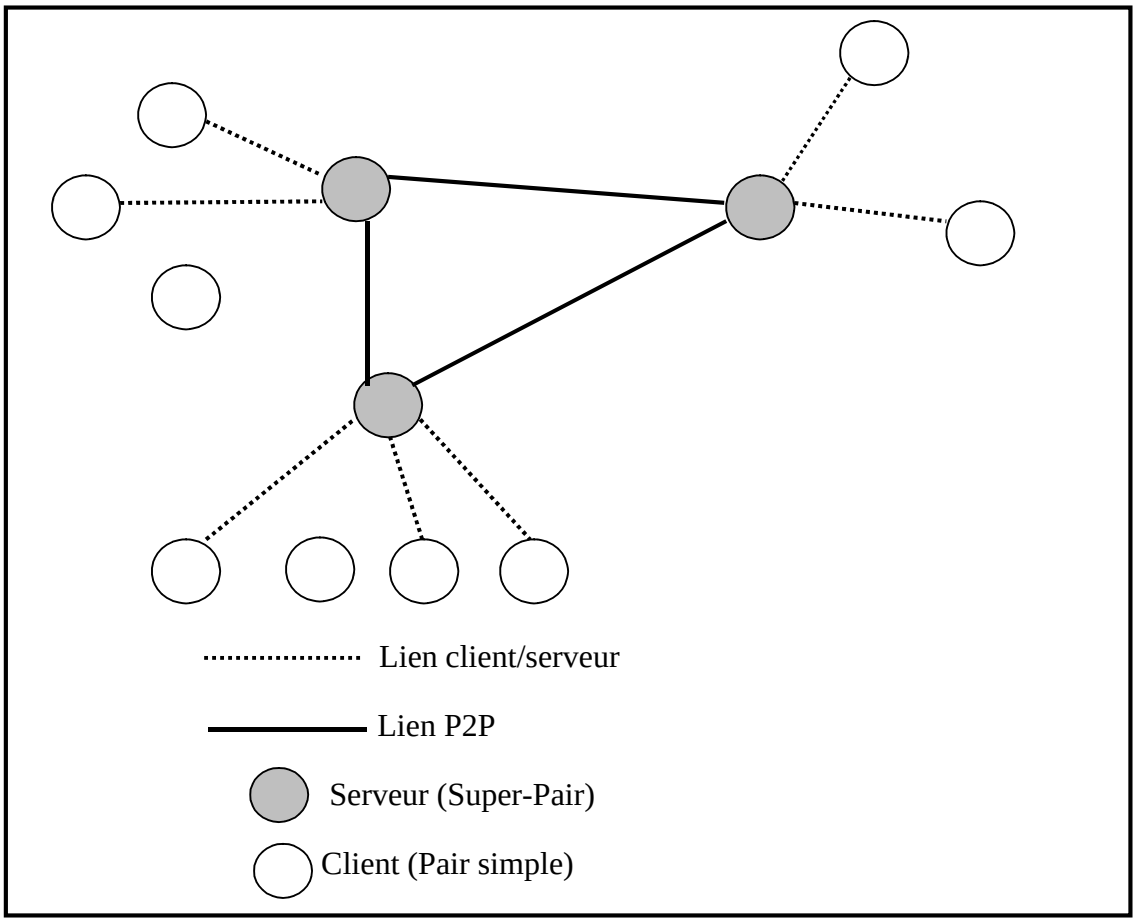


Figure1.4: P2P Hybride ou SuperPair.

Nous distinguons deux types de réseaux hybrides:

Les hybrides statiques : les super-pairs sont manuellement désignés dés le début. Ces derniers peuvent jouer en même temps le rôle du client comme on le retrouve dans l'exemple du réseau **eDonkey [eDonkey]**.

Les hybrides dynamiques : Dans certaines conditions, le logiciel client décide de transformer un nœud client en super-pair, comme dans l'exemple de Kazaa **[kazaa]**.

6. Etude comparative des architectures P2P

Chaque type d'architecture P2P, présenté précédemment, possède des avantages et des inconvénients par apport aux objectifs initiaux des systèmes P2P. Nous avons établi une étude comparative basée sur des caractéristiques correspondant aux typologies des réseaux P2P.

	Centralisés	**Non structurés**	**Structurés**	**Hybrides**
Partage et réduction des coûts	Fort	Moyenne	Moyenne	Fort
dynamisme	Faible	Fort	Moyenne	Moyenne
Décentralisation	Faible	Très forte	Très forte	Moyenne
Autonomie	Faible	Très forte	Très forte	Moyenne
Anonymat	Faible	Moyenne	Faible	Moyenne
Passage à l'échelle	Très faible	Faible	Fort	Fort
Tolérance aux pannes	Très faible (1 seul serveur)	Très fort	Moyenne	Moyenne
Orientation vers un autre modèle	Non	Oui (hybride)	Non	Non
Protocoles utilisés	?	Inondation	DHT	?

Tableau 1.1: Comparaison entre les architectures P2P.

Les avantages de l'architecture centralisée sont la facilité d'implémentation et d'administration ainsi que la rapidité et l'efficacité de la recherche. Alors que, ce type d'architecture soufre de plusieurs problèmes, l'inconvénient principal est la présence d'un seul point de faille dans le système (le serveur central), ce qui rend ces systèmes fragiles aux pannes, à la surveillance (absence d'anonymat), et aux attaques. De plus, ces systèmes offrent une faible performance de passage à l'échelle principalement due à la limite de la taille de la base de données du serveur et sa capacité à répondre aux requêtes (saturation de la bande passante du serveur).

Les systèmes non structurés présentent des limites au niveau du parcours du réseau : une requête peut être stoppée par une expiration du TTL (Time To Live) sans parcourir l'intégralité du réseau. Un autre problème réside dans l'importance de la bande passante consommée par les broadcastes générés par les requêtes de recherche.

Le point fort de ce type de réseau est la persistance aux pannes grâce à l'autonomie des nœuds. En effet, il est impossible d'arrêter le fonctionnement du réseau parce qu'il n'existe pas un serveur central.

D'un point de vue réduction du temps de recherche, les réseaux structurés présentent un grand avantage. En effet, la fonction de recherche minimise le nombre de sauts, le mécanisme de recherche permet d'accéder à n'importe quelle donnée en log(n) sauts. Par contre, selon les opérations liées au partage de l'index, la procédure d'entrée dans le réseau pour un nouveau nœud est plus complexe. De plus, la plupart des protocoles structurés sont sensibles aux pannes, à cause de la notion de successeur implémenté par la DHT.

Les réseaux hybrides héritent de quelques avantages des systèmes centralisés et ceux décentralisés avec certains inconvénients de ces systèmes. La tolérance aux pannes est un problème sérieux dans ce type de réseau, surtout ceux qui sont statiques. De plus, l'administration des clusters est une tâche délicate dans le cas où un pair peut joindre plusieurs groupes en même temps.

7. Conclusion

Dans ce chapitre, nous avons présenté l'un des types d'applications distribuées appartenant à la deuxième génération des architectures réparties. Pratiquement, après le grand succès de la première génération présentée par l'architecture client/serveur, cette dernière a rencontré plusieurs problèmes liés beaucoup plus à la centralisation des ressources de stockage et de traitement dans la partie serveur.

Pour régler les problèmes de l'architecture client/serveur et pour assurer des nouvelles fonctionnalités au niveau des applications distribuées, le paradigme P2P a vu le jour. Il est basé sur l'égalité des nœuds d'un réseau, où tous les pairs jouent le même rôle dans une architecture plate, chacun d'entre eux peut être client et/ ou serveur en même temps.

Cette caractéristique de base a permis aux systèmes P2P d'être impliqués dans plusieurs types d'applications distribuées : calcul distribué, diffusion de contenu, collaborations, moteurs de recherches et plateformes.

Les systèmes P2P sont structurés selon différentes architectures. Leur première génération été centralisée avec le premier logiciel P2P Napster. Ensuite, d'autres modèles, totalement décentralisés, ont été proposés. Ce type d'architecture est lui-même divisé en réseaux P2P non structurés et structurés. La combinaison entre les architectures centralisées et décentralisées est connue sous le nom des réseaux « hybride ».

Chacune de ces architectures présente des avantages, des inconvénients et des caractéristiques qui permettent aux systèmes P2P d'être plus ou moins adaptée à un type particulier d'applications suivant les objectifs visés par ces dernières.

Afin de terminer la présentation des concepts de base liés à notre travail, dans le chapitre suivant, nous allons présenter l'architecture orientée services, les Web services et ses technologies adjacentes, la convergence entre les Web services et le Web sémantique et enfin les différentes méthodes de découverte des Web services.

Chapitre 2

SOA, Web services, Web sémantique et découverte des Web services

1. Introduction

La diversité des applications d'un même système d'information, et la nécessité de manipuler des quantités de plus en plus importantes de données, posent un grand problème d'intégration d'applications. En effet, la communication entre les applications et les différents systèmes d'information devient de plus en plus difficile à cause des technologies hétérogènes utilisées qui amènent les utilisateurs à travailler dans un environnement incohérent, mal adapté et incompatible. Le manque d'interopérabilité est alors relevé comme principal problème de ces systèmes.

Dans ce contexte, et afin de résoudre ces problèmes, le concept d'EAI est apparu comme solution. Cependant, le monde d'EAI est très vaste car il englobe plusieurs approches, techniques et technologies d'intégrations qui ont évolué dans le temps depuis que le concept d'EAI a vu le jour. Actuellement, les solutions EAI de la deuxième génération sont orientées vers les processus métiers et les échanges interentreprises de sorte qu'elles prennent en compte les nouveaux modèles économiques créés et promus par Internet et ses technologies (TCP/IP, SMTP, HTTP, FTP, XML, etc.). De plus, les progiciels de gestion intégrés de la deuxième génération apportent une quantité de nouveaux modules organisés autour d'une nouvelle vision pour les systèmes d'information. Le but est de créer un nouveau modèle de collaboration, offrant une interopérabilité entre les systèmes d'information ; c'est là l'objectif de l'architecture SOA considérée, actuellement, comme la solution adéquate aux problèmes d'intégration des systèmes d'information. Cette architecture est urbanisée sous la forme de services réutilisables qu'il est possible de découvrir et composer dynamiquement, avec un couplage faible.

Pour répondre aux enjeux de la SOA, les Web services constituent une des technologies actuelles la mieux placée pour mettre en place l'architecture orientée services. Les Web services, qui sont basés sur des technologies Web dérivées du fameux standard XML, présentent de nombreux atouts pour faire communiquer des systèmes caractérisés par une hétérogénéité croissante. Ils permettent également de mettre en œuvre des services applicatifs partagés et de gérer la connectivité aux données. Ils sont devenus la technologie la plus utilisée pour l'interopérabilité et l'intégration des applications et des systèmes d'information.

Dans cette perception, la composition des Web services est devenue une solution adéquate pour implémenter les processus métiers fortement distribués et pour répondre, d'une manière efficace et rapide, aux besoins aux requêtes des internautes.

Cependant, le haut degré d'interopérabilité technique fournie par les Web services instaure le besoin d'introduire la sémantique dans cette technologie afin d'automatiser les différentes phases de leur cycle de vie, tout particulièrement la phase de découverte. La technologie du Web sémantique se présente alors comme une solution pour faciliter la découverte et la manipulation automatique (par ordinateur) des Web services et la naissance du concept des Web services sémantiques comme fruit de la convergence entre les Web services et le Web sémantique. En effet, son ultime objectif est de rendre les Web services plus accessibles à la machine en automatisant les différentes tâches qui facilitent leur utilisation.

Ce chapitre comportera deux parties. Dans la première, nous présentons l'architecture orientée services et ses différentes caractéristiques. Nous exposons aussi la notion de Web services et ses concepts adjacents : technologies de base et composition. Après, nous fournissons les caractéristiques du Web sémantique et la convergence entre ce dernier et les Web services. Ensuite, nous décrivons les différents langages de description sémantique des Web services. Dans la deuxième partie, nous présentons la problématique de la découverte des Web services, ainsi que quelques travaux de recherches qui décrivent des architectures, basées sur les systèmes P2P, conçues pour la découverte des Web services sémantiques.

PARTIE 1 : SOA, Web services et Web Sémantique

2. Caractéristiques de l'Architecture SOA

2.1 Historique et définition

L'architecture logicielle est une pratique relativement nouvelle dans le domaine du génie logiciel. Elle décrit les composants du système et la manière dont ces éléments interagissent à un niveau élevé. Les interactions entre les composants s'appellent les connecteurs, la configuration des composants et les connecteurs fournissent une vue structurale et comportementale du système **[STE09]**.

2.1.1 Histoire de la SOA

SOA n'est pas une solution, c'est une pratique. Le concept SOA a été inventé la première fois par l'analyste Gartner Yefim V. Natis en 1994 **[NAT+94]**. Selon Natis: «*SOA est une architecture de logiciel qui débute avec une définition d'interface et la construction entière d'application de topologie comme topologie des interfaces, des réalisations d'interface, et des appels d'interface…*» **[CHR+08]**.

2.1.2 Définition

SOA est un style architectural qui permet de construire des solutions d'entreprises basées sur les services. Ces derniers rassemblent les grandes applications de l'entreprise (dites applications composites) en **services interopérables** et **réutilisables [CRO05], [ROS+08]**.

Le service un composant logiciel exécuté par un **fournisseur** (Provider) à l'attention d'un **client** (Requester). Cependant, l'interaction entre un client et un fournisseur est matérialisée grâce à un protocole responsable de l'échange des messages entre les deux entités.

L'objectif d'une architecture orientée services est donc de décomposer les fonctionnalités d'un système ou d'une application en un ensemble de fonctions basiques, appelées **services**, implémentés par des composants, et de décrire les différentes interactions possibles entre ces services. L'objectif de cette opération est de créer une architecture logicielle globale décomposée en services correspondant aux processus métiers de l'entreprise. De ce fait, les applications d'un système d'information seront vues comme une collection de services qui

interagissent et communiquent entre eux. Cette communication peut consister en un simple retour de données ou en une activité plus complexe (coordination de plusieurs services).

Cependant, l'aspect le plus important de l'architecture SOA est qu'elle permet de séparer l'implémentation du service de son interface. C'est cette caractéristique qui assure le haut degré d'interopérabilité visé par cette architecture.

2.2 Concepts de SOA

SOA est plus qu'un ensemble de technologies. Elle n'est directement liée à aucune technologie, bien qu'elle soit le plus souvent mise en application avec des Web Services qui sont considérés comme la technologie la plus appropriée pour la réalisation de SOA. Cependant, l'utilisation des Web Services n'est pas proportionnée pour construire SOA. Nous devons employer les Web Services selon les concepts que SOA définit **[JUR+06]**. Les concepts de SOA les plus importants sont :

- ✓ Services
- ✓ Interfaces Self-describing (description d'individu)
- ✓ Échange des messages
- ✓ Soutien de liaison synchrone et asynchrone
- ✓ Accouplement faible
- ✓ Enregistrement de service
- ✓ Qualité du service
- ✓ Composition des services dans des processus d'affaires

2.3 Composants de la SOA

Afin de déduire les composants de bases de l'architecture SOA, il est nécessaire de présenter en premier lieu son paradigme de fonctionnement.

Le paradigme "découvrir, interagir et exécuter" comme montré dans la figure 2.1 permet au consommateur du service (client) d'interroger un **annuaire** pour le service qui répond à ses critères. Si l'annuaire possède un tel service, alors il renvoie au client le **contrat** du service voulu ainsi que son adresse. SOA consiste en quatre entités configurées ensemble pour supporter le paradigme découvrir, interagir et exécuter **[STE02]**.

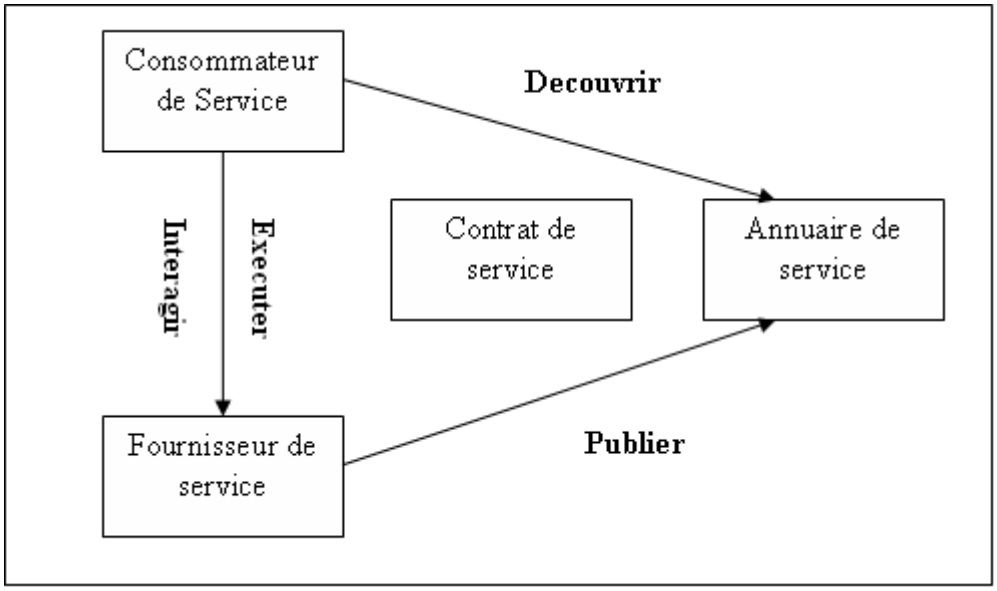


Figure 2.1 Paradigme "découvrir, interagir et exécuter".

Suivant ce protocole de fonctionnement, nous pouvons présenter les composants de l'architecture SOA comme suit **[STE02]**:

2.3.1 Le consommateur de service

Le consommateur de service est une application qui requière un service. C'est l'entité qui initie la localisation du service dans l'annuaire, interagit avec le service à travers un protocole et exécute la fonction exposée par le service.

2.3.2 Le fournisseur de service

Le fournisseur de service est une entité adressable via un réseau, il accepte et exécute les requêtes venant d'un client.

Le fournisseur de service publie le contrat de service dans l'annuaire pour qu'il puisse être accédé par les clients.

2.3.3 L'annuaire de service

L'annuaire de service est un annuaire qui contient les services disponibles. C'est une entité qui accepte et sauvegarde les contrats du fournisseur de service et présente ces contrats aux éventuels clients.

2.3.4 Le contrat de service

Le contrat spécifie la manière dont le client de service va interagir avec le fournisseur de service. Il spécifie le format de la requête et la réponse du service.

3. Web services

D'après la définition, SOA est une approche architecturale qui ne fait aucune hypothèse sur la technologie de mise en œuvre. En particulier, l'amalgame souvent faite entre SOA et les Web services est une erreur **[FOU+06]**.

Cependant, la conception des spécifications Web services a été menée dans l'objectif de répondre au mieux aux enjeux de l'architecture SOA **[FOU+06]**. Les Web services fournissent les bases technologiques nécessaires à la réalisation de l'interopérabilité entre les applications en utilisant différentes plateformes, différents systèmes d'exploitation et différents langages de programmation **[POT+03]**.

3.1 Définition

Un Web service est un composant logiciel identifié par une URI, qui possède une interface publiable. Cette dernière peut être découverte par d'autres systèmes, qu'ils peuvent interagir avec le Web service selon les règles prescrite par sa description, en utilisant des messages basés sur XML et portés par des protocoles standards d'internet.

Les Web Services fournissent une couche d'abstraction entre le client et le fournisseur d'un service. Cette couche est indépendante de la plateforme et du langage d'implémentation **[HAM+03]**, grâce à un ensemble de protocoles standardisés comme **SOAP** (Simple Object Access Protocol), **WSDL** (Web Service Description Language) et **UDDI** (Universal Description, Discovery and Integration) **[BEL09]**.

Cette définition implique que les Web services :

- sont des composants d'application ;
- communiquent en utilisant des protocoles standards ;
- sont autonomes et auto- descriptifs ;
- peuvent être découverts ;
- peuvent être utilisés par d'autres applications ;
- s'appuient sur XML (Extensible Markup Language) ; et enfin
- sont extensibles : chacun peut adjoindre ses propres données, protocoles ou mécanismes propriétaires.

3.2 Fonctionnement des Web services

Dans sa première génération, le fonctionnement des Web Services repose sur trois couches fondamentales présentées comme suit:

- **Invocation** : visant à établir la communication entre le client et le fournisseur en décrivant la structure des messages échangés.
- **Découverte :** permettant de localiser un Web service particulier dans un annuaire de services décrivant les fournisseurs ainsi les services fournis.
- **Description :** dont l'objectif est la description des interfaces des Web services (paramètres des fonctions, types de données).

Pour assurer ces fonctionnalités, trois technologies de base ont été proposées et disposées en couches pour construire l'architecture de base des Web Services, comme le montre dans la table suivante.

UDDI	Découverte de services
WSDL	Description de services
SOAP	Communications
XML	

Table 2.1 : Couches technologiques des Web Services [VEZ05].

Les couches XML et SOAP sont les couches de plus bas niveau, elles permettent le transport de l'information alors que WSDL permet de décrire le service aux utilisateurs externes. Enfin la couche la plus haute UDDI décrit ce que peut produire le service, c'est la couche la plus sémantique **[VEZ05]**.

Ces technologies sont présentées dans la section suivante.

3.3 Infrastructure des Web services

3.3.1 XML

XML constitue la technologie de base des architectures Web services ; c'est un facteur important pour contourner les barrières techniques. XML est un standard qui permet de décrire des documents structurés transportables sur les protocoles d'Internet. En effet, il apporte à l'architecture des Web services l'extensibilité et la neutralité vis à vis des plates-formes et des langages de développement.

La technologie des Web services a été conçue pour fonctionner dans des environnements totalement hétérogènes. Cependant, l'interopérabilité entre les systèmes hétérogènes demande des mécanismes puissants de correspondance et de gestion des types de données des

messages entre les différents participants (clients et fournisseurs). C'est une tâche où les schémas de type de données XML s'avèrent bien adaptés. C'est pour cette raison que la technologie des Web services est essentiellement basée sur XML ainsi que les différentes spécifications qui tournent autour (les espaces de nom, les schémas XML, et les schémas de Type) **[MEL04]**.

3.3.2 Communication avec SOAP

SOAP est un protocole basé sur XML qui permet l'échange léger de données structurées entre des applications exécutées sur différents systèmes d'exploitation, avec différentes technologies et différents langages de programmation. Il peut être employé dans tous les styles de communication : synchrones ou asynchrones, point à point ou multi-points. Il se contente d'offrir la possibilité de structurer des messages destinés à des objectifs particuliers. Néanmoins, il est particulièrement utile pour exécuter des dialogues requête-réponse RPC (Remote Procedure Call) **[HAM+03] [MEL04] [Soap]** bien que RPC présente un problème de compatibilité et de sécurité de sorte que les pare-feux et les serveurs proxy vont normalement bloquer ce genre de trafic. Une meilleure façon de communiquer entre les applications est en utilisant http qui est accepté par tous les navigateurs Web ainsi que tous les serveurs. SOAP a été principalement créé pour atteindre ce but.

Un message SOAP est un document XML ordinaire qui contient les éléments suivants :

- L'élément *Envelope* qui identifie le document XML comme étant un message SOAP
- L'élément *Header* qui est optionnel et qui contient des informations d'entête
- L'élément *Body* qui contient l'appel ainsi que la réponse retournée
- L'élément *Fault* qui est optionnel et qui fournit des informations sur d'éventuelles erreurs survenues lors de l'analyse du message

Tous ces éléments cité ci-dessus sont déclares dans les *namespace* de l'enveloppe SOAP :

"http://www. w3.org/2001/12/soap-envelop"

Et le *namespace* pour le SOAP encoding et les types de données :

http://www.w3.org/2001/12/soap-encoding

3.3.3 Description des Web services

Le langage WSDL (Web Service Description Language [CHR+01]) proposé par Ariba, IBM et Microsoft auprès du W3C est un format de description des Web services fondé sur XML. Il permet de décrire de façon précise les Web services en incluant des détails tels que

les protocoles, les serveurs, les ports utilisés, les opérations pouvant être effectuées, les formats des messages d'entrée et de sortie ainsi que les exceptions pouvant être renvoyées. Il permet la représentation d'un Web Service de manière plus abstraite pour la réutilisation **[HAM+03].** Il est devenu une recommandation du W3C depuis le 26 Juin 2007.

Un document WSDL contient des informations opérationnelles concernant le service. La définition du service marquée par la balise **<definition**s> est la racine de tout document WSDL. Elle peut contenir les attributs précisant le nom du service et les espaces de nommage. Un document WSDL contient les entités suivantes:

- **Types:** précise les types de données complexes, pour lequel WSDL emplois XML Schéma.
- **Message:** l'abstraction décrivant les données échangées entre Web services.
- **Opération:** l'abstraction décrivant une action implémentée par un Web service.
- **Type de port:** un ensemble d'opérations implémenté par une terminaison donnée.
- **Liaison (binding):** un protocole concret d'accès à un port et un format de spécification des messages et des données pour ce port.
- **Port:** un point de terminaison identifié de manière unique par la combinaison d'une adresse Internet et d'une liaison.
- **Web Service:** une collection de ports.

Il est important de mentionner que le document WSDL est divisé en deux parties : l'interface du service et son implémentation. L'interface du service est la partie réutilisable de la définition du service, elle peut être référencée par de multiples implémentations du service. Elle est composée des balises : Types, Message, Opération et Liaison. Alors que la partie implémentation, décrite par les balises Port et Service, est unique et présente une terminaison pour invoquer le Web service. De plus, chaque document WSDL peut être documenté grâce à une balise **<documentation>** bien que cet élément soit facultatif.

3.3.4 Découverte des Web services

UDDI (Universal Description, Discovery and Integration) est une spécification pour la description et la découverte de Web Services en utilisant XML Schéma. Il convient alors de décrire les quatre constitutions de l'enregistrement d'un Web service: l'identité du fournisseur (pages blanches), la description du ou des services offerts (pages jaunes), l'information sur les modes d'exploitation et les différents modèles de données sous-jacents **[CHA02]**. Ainsi,

UDDI se présente comme un ensemble de bases de données utilisées par les entreprises pour enregistrer leurs Web services ou pour localiser d'autres Web services.

Grâce à UDDI, les entreprises peuvent enregistrer des données les concernant, des renseignements sur les services qu'elles offrent et des informations techniques sur le mode d'accès à ces services. Une fois l'enregistrement terminé, les informations sont automatiquement répliquées sur l'ensemble des annuaires. Ce fonctionnement permet aux services d'être découverts par un plus grand nombre d'entreprises.

UDDI offre deux fonctionnalités au sein de l'architecture globale : la publication et la découverte. Elles permettent aux fournisseurs de publier leurs services selon un modèle de description et au client l'interrogation des services. De ce fait, la spécification UDDI constitue une norme pour les annuaires des Web services. Les fournisseurs disposent d'un schéma de description permettant de publier des données concernant leurs activités, la liste des services qu'ils offrent et les détails techniques sur chaque service. De plus, la norme UDDI offre aussi une API aux applications clientes, pour consulter et extraire des données concernant un service et/ou son fournisseur **[MEL04]**.

3.4 Composition des Web services

Dans cette section, nous présentons la composition des Web services et ses concepts adjacents.

3.4.1 Définition

«Les composants sont destinés à être composés» **[Bro96]**. Selon cette définition, la réutilisation est un avantage important de l'approche par composants **[Szy97]**, **[PFI+96]**. Par définition, les Web services, tels qu'ils sont présentés, sont des composants conceptuellement limités à des fonctionnalités relativement simples qui sont modélisées par une collection d'opérations **[MEL04]**. De ce fait, les Web services doivent pouvoir être composés en services que l'on compose jusqu'à ce que le service résultant fournisse un support entier pour les processus métiers.

La composition des Web services a été définie par **[CAS+02]** comme étant la capacité d'offrir des services à valeur ajoutée en combinant des services existants probablement offerts par différentes organisations. Techniquement parlant, la composition permet d'assembler des Web services afin d'atteindre un objectif particulier, par l'intermédiaire de primitives de contrôles (boucles, test, traitement d'exception, *etc.*) et d'échange (envoi et réception de

messages) **[KEL+06]**. Il s'agit en d'autres termes de spécifier les services qui seront invoqués, dans quel ordre et comment gérer les conditions d'exceptions **[BOU09]**.

3.4.2 Orchestration et Chorégraphie

La composition des Web services peut être faite en utilisant l'une des deux méthodes : orchestration ou chorégraphie.

Dans la technique de l'orchestration, un processus central (qui peut être lui-même un Web service) prend le contrôle de tous les Web services impliqués dans la composition et coordonne l'exécution des différentes opérations sur ces Web services. Ces derniers ne savent pas (et n'ont pas à savoir) qu'ils sont invoqués dans une composition et qu'ils font partie d'un processus métier complexe. Seul le coordinateur central de l'orchestration le sait. L'orchestration est donc centralisée avec des définitions explicites des opérations et l'ordre d'invocation des Web services **[JUR+06].**

D'autre part on peut ne pas compter sur un coordinateur central. Au lieu de cela, chaque Web service impliqué dans la chorégraphie sait exactement quand exécuter ses opérations et avec qui interagir. La chorégraphie est un effort de collaboration focalisé sur l'échange de messages dans des processus métiers publics. Tous les participants à la chorégraphie doivent connaître leurs rôles dans le processus métier, les opérations à exécuter, les messages à échanger, et le temps d'échange de ces messages. **[JUR+06]**.

Du point de vue de la composition des Web services, pour exécuter des processus métiers, l'orchestration est avantageuse par rapport à la chorégraphie:

- On connaît de façon exacte qui est le responsable de l'exécution du processus métier en entier.
- On peut incorporer les Web services, même ceux qui ne savent pas qui sont impliqués dans des processus métiers.
- On peut aussi fournir un scénario alternatif en cas d'erreurs.

3.4.3 BPEL pour la composition des Web services

Plusieurs initiatives ont été inventées pour automatiser la composition des Web services. Cependant le langage BPEL (Business Process Execution Langage) présente la technique la utilisée par les développeurs des processus métiers.

a) Besoin d'un langage spécifique à la composition des Web services

La composition des Web services est définie comme étant un processus métier où les services présentent une collection d'activités qui collaborent pour implémenter ce processus.

Pour une vue extérieure, le processus métier résultant est un Web service (composite) vu comme n'importe quel autre service basique. Par conséquent, les Web services composites doivent être considérés récursivement comme des Web services et doivent garder les mêmes caractéristiques que les services basiques (services WSDL) à savoir auto-descriptifs, interopérables, et facilement intégrables **[MEL04]**.

La mise en place d'un processus métier à travers la composition des Web services nécessite la définition de l'ensemble des services participants ainsi que l'échange des messages entre ces services. Dans ce contexte, les interactions peuvent être sans états, synchrones, asynchrones, comme elles peuvent appartenir à un état qui combine tous ces modes en même temps. De plus, les compositions complexes de plusieurs Web services consistent souvent en des échanges de messages dans un ordre bien défini. Ainsi que les interactions sont généralement assez longues. D'autre part, un autre aspect important est la capacité de décrire la façon dont les erreurs sont traitées **[JUR+06]**.

Pour cette raison, le langage WSDL est insuffisant pour décrire les compositions complexes des Web services. WSDL fournit une description basique et une spécification des messages échangés, mais cette description ne permet que de décrire de simples interactions entre le client et le Web service. Etant donné les limitations de WSDL, il est nécessaire d'avoir un mécanisme pour décrire la composition des Web services en des processus plus complexes. L'utilisation de technologies dédiées à la composition est une chose indispensable **[JUR+06]**.

b) Présentation de BPEL

Dans ce contexte, plusieurs initiatives ont vu le jour pour standardiser l'automatisation des processus métiers. BPEL représente le langage le plus accepté dans la communauté des concepteurs de processus métiers modernes.

Microsoft, IBM, et BEA ont développé la première version de BPEL en Aout 2002 et depuis que SAP et Siebel les ont rejoints, beaucoup de modifications et d'améliorations ont été apportées. En Avril 2003, BPEL a été soumis à l'OASIS (Organisation for the Advancement of Structured Information Standards) pour standardisation.

BPEL (connu aussi sous le nom BPEL4WS ou WSBPEL) représente la convergence de deux langages, WSFL (Web Services Flow Languages) et de XLANG. Il combine les deux approches et fournit un vocabulaire riche, basé sur XML, pour la description des processus métiers.

BPEL peut décrire des processus métiers aussi bien simples que complexes. Il offre des instrictions basiques comme : <**process**>, <**invoke**>, <**receive**>, <**reply**>, <**variable**>, <**assign**> et <**copy**> ainsi que des instructions structurées permettant de combiner les activités basiques. BPEL propose différentes activités structurées parmi lesquelles on retrouve :

- Définir un ensemble d'activités qui seront invoquées dans une séquence ordonnée en utilisant <**sequence**>
- Définir un ensemble d'activités qui seront invoquées en parallèle en utilisant <**flow**>
- Définir les structures conditionnelles en utilisant l'activité <**if**> <**else**>

En utilisant cette variété de constructeurs, BPEL nous permet de définir des processus métiers de manière algorithmique. Par conséquent cette définition est relativement simplifiée. De plus, BPEL facilite l'invocation des opérations des Web services, qu'elles soient synchrones ou asynchrones, et fournit un vocabulaire riche pour le traitement des erreurs.

4. Web sémantique

Avant de décrire la notion de Web services sémantiques, il est important de présenter le Web sémantique et ses concepts adjacents.

4.1 Origine

La théorie de l'information distingue fondamentalement trois niveaux pour la représentation du monde : la syntaxe, la sémantique et la pragmatique. La sémantique est généralement utilisée pour compléter la syntaxe qui constitue un élément nécessaire pour pouvoir parler de sémantique.

D'un point de vue informatique, les définitions des notions de sémantique et de syntaxe restent pratiquement similaires à celles pratiquées en linguistique : la syntaxe se réfère au respect de la grammaire formelle d'un langage, alors que la sémantique concerne plutôt l'interprétation des modèles et des symboles informatiques **[IZZ06]**. Cependant, le concept « sémantique » a émergé avec l'évolution du World Wide Web au **Web sémantique [BER+01]** où on la considère comme "une forme formelle de représentation des connaissances humaines" **[WOO75]**.

Au cours de la première décennie de son existence, la plupart des informations sur le Web sont conçues uniquement pour la consommation humaine. Les humains peuvent lire les pages Web et les comprendre, mais leurs significations intrinsèques ne sont pas claires pour permettre leur interprétation par des ordinateurs.

De nos jours, ce problème constitue le plus grand obstacle pour fournir un meilleur support aux utilisateurs du Web. Pour le moment, le sens du contenu du Web n'est pas "traitable par ordinateur", même s'il existe quelques outils pour retrouver du texte et faire des traitements sur ce texte, mais quand il s'agit d'interpréter son sens ou d'essayer d'extraire des informations utiles, la capacité des applications actuelles reste toujours limitée **[ANT+O8]**.

De ce fait, l'information sur le Web doit être définie de manière qu'elle puisse être utilisée par les ordinateurs, non seulement à des affichages, mais aussi pour l'interopérabilité et l'intégration entre les systèmes et les applications. Une façon de permettre l'échange machine à machine et le traitement automatisé est de fournir les informations de telle sorte que les ordinateurs peuvent comprendre. C'est précisément l'objectif du Web sémantique, pour rendre possible la transformation de l'information par les ordinateurs.

4.2 Définition du Web sémantique

Le « Web sémantique » est un terme inventé par **Tim Berners-Lee** qui en dit : *« Le Web sémantique est une extension du Web actuel, dans lequel l'information a un sens bien défini, et permet une meilleure coopération dans le travail entre les humains et les ordinateurs…Le Web des données qui peuvent être traitées par les ordinateurs »* **[YU07]**.

Le Web sémantique est donc une nouvelle approche pour l'organisation du contenu du Web instaurée dans le but d'améliorer l'interopérabilité, la découverte et la récupération de ressources.

Cependant, le Web sémantique n'est pas simplement dédié au World Wide Web. Il représente un ensemble de technologies qui fonctionneront également sur des Intranets d'une entreprise. Ainsi, le Web sémantique résoudra plusieurs problèmes principaux posés à des architectures courantes de technologie de l'information.

4.3 Architecture du Web sémantique

Le Web Sémantique n'est pas seulement une vision de l'avenir du Web. Certaines technologies sont déjà disponibles et figurent dans l'architecture illustrée dans la figure 2.2.

L'architecture repose sur des technologies de structuration de documents fondées sur XML. Sur cette base, des langages de métadonnées sémantiques de haut niveau ont été développés et permettent la description des ressources sur le Web. Afin de fournir un cadre interprétatif à ces métadonnées sémantiques le Web Sémantique utilise des ontologies. Au niveau supérieur, le raisonnement sur les données est assuré par des mécanismes d'inférence,

qui permettent d'une part de construire de la connaissance, mais aussi d'en maintenir la cohérence.

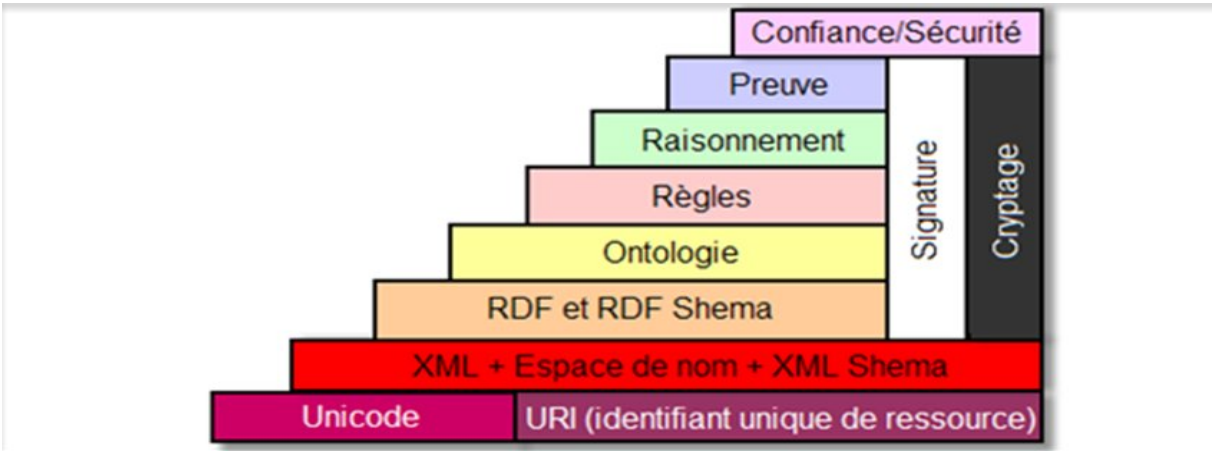


Figure 2.2 : Architecture du Web sémantique [BER+01].

Au niveau le plus haut, nous retrouvons la notion de confiance dont la principale question est : « Sur quelles bases peut on considérer comme vraie l'information disponible sur le Web?». Les langages et outils développés dans le cadre du Web sémantique ne sont que des bases technologiques. Ils ne résolvent pas à eux seuls les problèmes de présentation et d'utilisation de ces données sémantiques **[ANT+O8]**.

Dans les sections suivantes, nous présentons les concepts les plus importants liés au Web sémantique.

4.3.1 Métadonnées

En général, le terme "métadonnées" est utilisé pour désigner "des données à propos d'autres données", c'est à dire, des données qui décrivent des ressources d'information. Plus précisément, les métadonnées sont une méthode structurée pour décrire des ressources et par conséquent pour améliorer leur sens. Le terme structuré est important car des données structurées impliquent une lisibilité et une compréhensibilité par les ordinateurs **[YU07]**.

4.3.2 Description des ressources : RDF et RDFS

Le langage RDF (Ressource Description Framework) **[RDF]** a été développé par W3C comme langage basé sur les réseaux sémantiques pour décrire les ressources du Web. Les objectifs initiaux de RDF étaient la représentation de la sémantique et une meilleure exploitation des métadonnées. Mais, de manière plus générale, RDF permet de voir le Web comme un ensemble de ressources reliées par les liens « sémantiquement » étiquetés.

Les énoncés RDF sont des triplets ressource-attribut-valeur où la valeur est une ressource ou chaîne de caractères et une ressource doit disposer d'une URI. Les triplets sont interprétables comme sujet-prédicat-objet. La simplicité du modèle, critiquable pour certains, peut être une des clés de son acceptation et de la relative simplicité de la réalisation d'outils. Afin de renforcer ce langage, RDF Schema a été construit par W3C pour comme extension de RDF comportant des primitives basées sur des frames. RDF Schema permet notamment de déclarer les propriétés des ressources ainsi que le type de ces ressources. La combinaison de RDF et RDF Schema est connue sous le nom RDFS **[IZZ06], [LAU+03]**.

4.3.3 Ontologies

Indépendamment des définitions philosophiques du terme ontologie, en informatique il existe une multitude de définitions pour ce concept. La plus fréquemment référencée dans la bibliographie et la plus synthétique est celle de Gruber : « *une ontologie est une spécification formelle explicite d'une conceptualisation partagée*» **[GRU93]**. Le terme "conceptualisation" réfère à un modèle abstrait d'un certain phénomène de la réalité et qui permet d'identifier les concepts pertinents de ce phénomène. Le terme "explicite" signifie que le type des concepts utilisés ainsi que les contraintes sur leur emploi sont réellement définies d'une manière claire et précise **[IZZ06]**.

Pratiquement, une ontologie permet de décrire formellement un domaine de discours. Elle consiste en un ensemble fini de concepts et de relations entre ces concepts où un concept est une classe du domaine. Par exemple, dans une université les étudiants, les enseignants, le personnel constitue des concepts importants. Les relations constituent en particulier les hiérarchies entres les classes, une hiérarchie spécifie qu'une classe C est une sous-classe d'une autre classe C' si chaque objet de C est inclus dans la classe C' **[ANT+O8]**.

En plus des relations hiérarchiques, les ontologies peuvent inclure des informations telles que : les propriétés, les restrictions de valeurs, les déclarations de disjointure et les spécifications de relations logiques entre les objets.

Pour que les ontologies soient cruciales pour le Web sémantique, des méthodes et des outils sont développés pour contribuer à **[LAU+03]**:

- Construire les ontologies, que ce soit à partir de sources primaires, particulièrement les corpus textuels, ou en recherchant une certaine réutilisabilité.
- Gérer l'accès aux ontologies, leur évolution, avec gestion des versions, et leur fusion. Les ontologies sont souvent riches de plusieurs milliers de concepts et ne restent alors

directement appréhendables que par leur concepteur. Leur accès par des utilisateurs, mêmes professionnels, nécessite de gérer le lien entre les concepts des ontologies et les termes du langage naturel, que ce soit pour une simple compréhension ou pour l'indexation et la construction de requêtes destinées à des tâches de recherche d'information.

- Assurer l'interopérabilité des ontologies en gérant les hétérogénéités de représentations et les hétérogénéités sémantiques. Ces dernières sont les plus difficiles à gérer et nécessitent des réflexions conjointes à la problématique de l'accessibilité des ontologies.

4.3.4 OWL

Il existe plusieurs langages de spécification d'ontologies (ou langage d'ontologies) qui ont été développés pendant les dernières années, et qui deviendront sûrement des langages d'ontologie dans le contexte du Web sémantique. Cependant, ceux qui sont proposés par W3C restent les plus utilisés dans la communauté du Web sémantique.

Nous avons déjà présenté la proposition du W3C qui s'appuie au départ sur une pyramide de langages dont RDF et RDFS sont relativement stabilisées. RDF et RDFS permettent de définir, sous forme de graphes de triplets, des données ou des métadonnées. Cependant, de nombreuses limitations bornent la capacité d'expression des connaissances établies à l'aide de RDF/RDFS. On peut citer, par exemple, l'impossibilité de raisonner et de mener des raisonnements automatisés (automated reasoning) sur les modèles de connaissances établis à l'aide de RDF/RDFS ; c'est ce manque que se propose de combler OWL (Ontology Web Language).

Proposé par W3C, OWL **[OWL]** est un standard s'appuie sur le langage DAML+OIL, produit de la combinaison du langage américain DAML (Darpa Agent Markup Language) et OIL (Ontology Inference Layer) provenant de projets européens **[DAML]**. Le langage OWL est actuellement construit sur RDFS, et apporte ainsi aux langages du Web sémantique, l'équivalent d'une logique de description tout en disposant aussi d'une syntaxe XML **[LAU+03]**.

Il faut savoir que dans un langage pour écrire des ontologies il faut faire l'équilibre entre la capacité d'expression et l'efficacité du raisonnement, car plus le langage est expressif, plus l'efficacité du raisonnement diminue, et parfois le raisonnement devient tellement complexe qu'il n'est plus possible de le faire par ordinateur. Le but est alors de concevoir un langage

qui permet cet équilibre et c'est ce qui a motivé le W3C à prendre la décision de définir trois sous langages d'OWL **[YU07]**:

a) OWL Full : Comme son nom l'indique OWL Full contient toutes les constructions possibles avec le langage OWL.

b) OWL DL : C'est un sous ensemble de OWL Full qui suit les règles suivantes:

- Une classe ne peut être membre d'une autre classe.
- Des restrictions sont établies sur les propriétés fonctionnelles, fonctionnelles inverses, et transitives.
- Une restriction est prescrite sur OWL : imports, lorsqu'on développe avec OWL DL on ne peut importer un document écrit avec OWL Full.

c) **OWL Lite** : C'est un sous ensemble de OWL DL caractérisé par :

- owl :hasValue, owl :disjointWith, owl :unionOf, owl :complementOf et owl :oneOf ne sont pas autorisées.
- Les contraintes de cardinalités sont plus réduites.

5. Web services sémantiques

Le couplage entre les Web services et le Web sémantique s'inscrit dans le cadre d'intégration des systèmes hétérogènes, plus particulièrement l'intégration sémantique. Dans ce contexte, plusieurs approches ont été proposées comme l'intégration sémantique par les données, par les traitements et par les processus. Dans notre travail, nous nous intéressons à l'approche orientée services sémantiques comme l'une des approches les plus efficaces qui est basée à la fois sur le dynamisme et la sémantique (figure 2.2) **[CAR06], [IZZ06]**.

Les Web services sont devenus un moyen très efficace dans l'interopérabilité des systèmes. Le besoin d'introduire la sémantique dans les Web services se fait sentir, afin d'automatiser les différentes phases de leur cycle de vie, en l'occurrence la phase de découverte. Le concept des Web services sémantiques, est le fruit de la convergence du domaine des Web services avec le Web sémantique (voir la figure 2.3**, [FEN+02 a]**). En effet, son ultime objectif est de rendre les Web services plus accessibles à la machine en automatisant les différentes tâches qui facilitent leur utilisation.

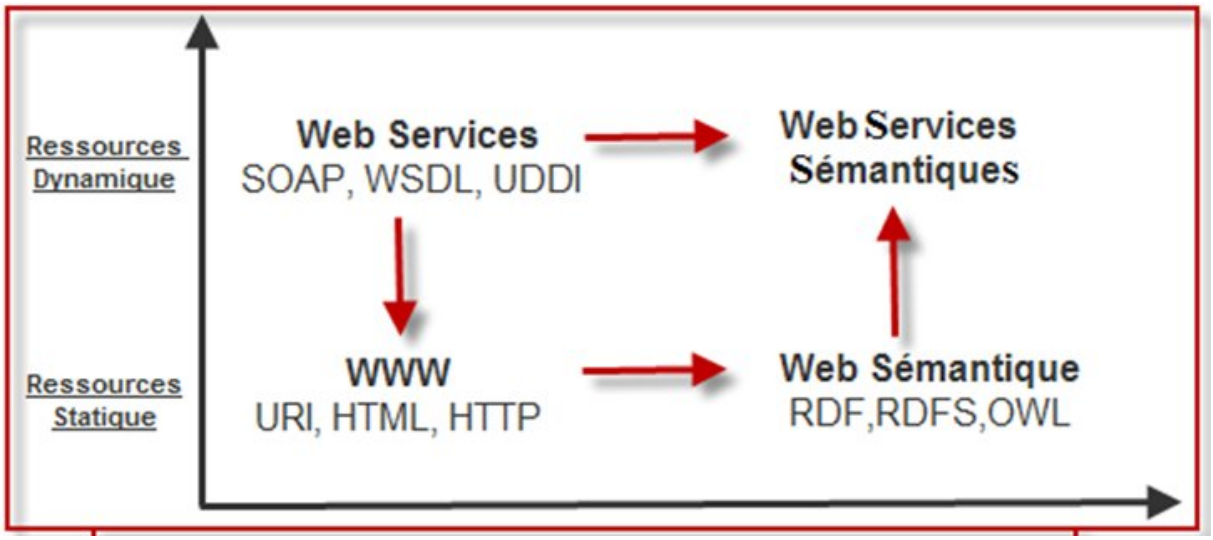


Figure 2.3 : Origine des Web services sémantiques [CAR06].

Dans cette section, nous présentons les approches les plus représentatives des Web services sémantiques, spécialement en ce qui est, à notre sens, le plus efface et le plus utilisé dans la communauté du Web sémantique à avoir OWL-S.

5.1 OWL-S

Sachant que différents Web services peuvent offrir différentes fonctionnalités, prendre différents paramètres d'entrées et différents paramètres de sorties, chacun de ces Web services peut être décrit en répondant à des questions telles que **[YU07]**:

- Quelle est la fonction que remplit ce Web service ?
- Comment fonctionne-t-il ?
- Comment peut-il être invoqué ?

Si on crée une ontologie qui nous permettra de répondre à ces questions générales, cela constituera aussi une ontologie générale sachant que les classes et les propriétés de cette ontologie ne doivent être liées à aucun domaine **[YU07]**.

Prédécesseur de DAML-S (Darpa Agent Markup language for Services), OWL-S **[OWL-S]** est une ontologie appelée *upper Ontology* qu'on appellera ontologie supérieure OWL-S pour (Web Ontology Language-Services), elle est écrite en utilisant OWL et a pour objectif la description des Web services **[YU07]**.

OWL-S : est une ontologie dédiée à la description des capacités et des propriétés des Web services. Le but d'OWL-S est de permettre l'automatisation de *la recherche*, de *la découverte*, de *l'invocation* et de *l'interconnexion* des Web services. Il fournit des éléments de description

et spécifie les relations entre ceux-ci ainsi qu'un modèle permettant de décrire les Web services en introduisant des informations sémantique et en séparant les fonctionnalités du service de son fonctionnement interne et de la façon d'y accéder.

L'objectif d'OWL-S est de répondre aux trois questions précédemment citées. Pour cela, une ontologie OWL-S est constituée de trois sous ontologies : *Profile.owl*, *Process.owl* et *grounding.owl* qui ont pour but de décrire un Web service. Pour relier ces trois sous-ontologies ensemble, OWL-S présente une ontologie de plus haut niveau final appelé l'ontologie de Service : *service.owl* **[YU07]**. Toutes ces sous-ontologies sont écrites en utilisant OWL et la figure 2.4 présente la description des Web services dans la mesure du OWL-S.

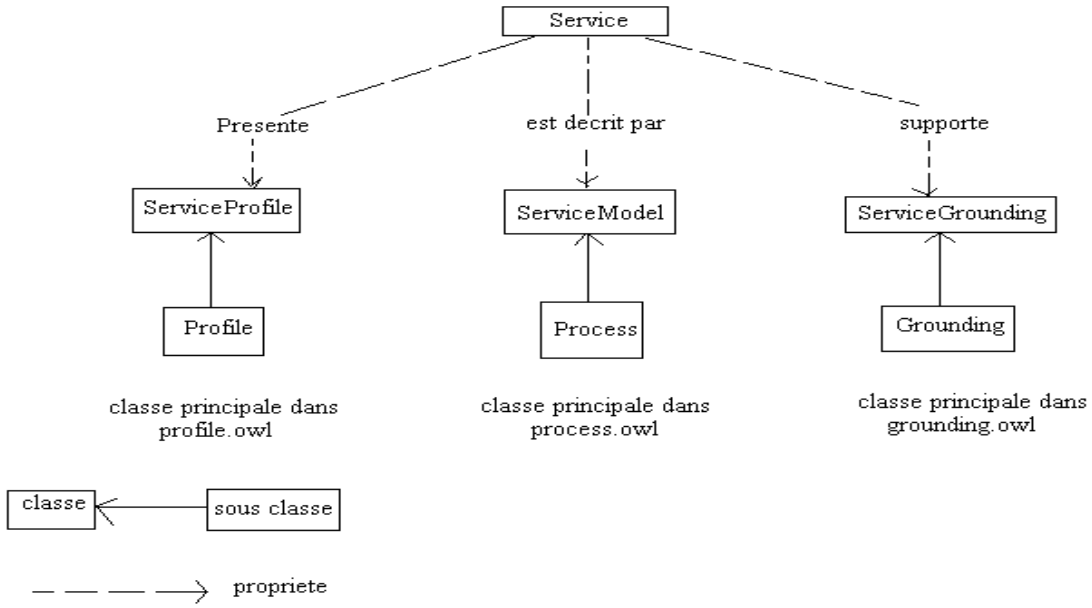


Figure 2.4 : Ontologie de haut niveau d'OWL-S [OWL-S].

Le rôle de chaque sous ontologies apparaissant dans la figure précédente est décrit comme suit **[YU07]**:

- **Fonctions d'un Web service**

L'ontologie supérieure OWL-S contient une sous ontologie appelée *Profile (profile.owl)* qui permet de définir des classes et des propriétés répondant à la question relative à la fonction du Web service. Cette ontologie a pour objectif la découverte du Web service. Elle sera donc utilisée lors de la recherche de ce dernier et c'est celle qui nous intéresse tout particulièrement.

- **Fonctionnement du Web service**

Une autre sous ontologie utilisée est *Process (process.owl).* Elle définit les classes et les propriétés pour pouvoir décrire comment fonctionne le Web service. Plus précisément, elle décrit les procédures nécessaires pour que le client puisse interagir avec le Web service.

- **Invocation du Web service**

Une autre sous ontologie est *Grounding (grounding.owl),* elle permet de savoir comment un demandeur peut techniquement accéder à un service (par exemple quel est le protocole utilisé lors de l'échange).

5.2 WSMO

WSMO (*Web Service Modeling Ontology*) **[BUS+04]** est un langage formel et une ontologie pour la description de divers aspects liés aux Web services sémantiques. Dans l'approche WSMO, on distingue deux composants principaux qui sont WSML (Web service Modeling Language) **[BRU+05]** et WSMX (Web Service Execution Environment) **[BUS+04]**.

WSML fournit un langage formel pour la description des éléments définis dans l'architecture WSMO. Il est basé sur différents langages logiques qui sont la logique de description, la logique de premier ordre et la logique de programmation. WSMX est un environnement d'exécution qui permet d'offrir un certain nombre de modules utilisables en temps d'exécution permettant de prendre en charge la découverte, la sélection, la médiation et l'invocation des services sémantiques **[IZZ06]**.

WSMO définit les éléments de modélisation pour la description de plusieurs aspects des Web services sémantique fondés sur l'échouement conceptuel mis en place dans le cadre de la modélisation des Web services **[FEN+02 b]**. Quatre composantes principales sont définies (figure 2.5):

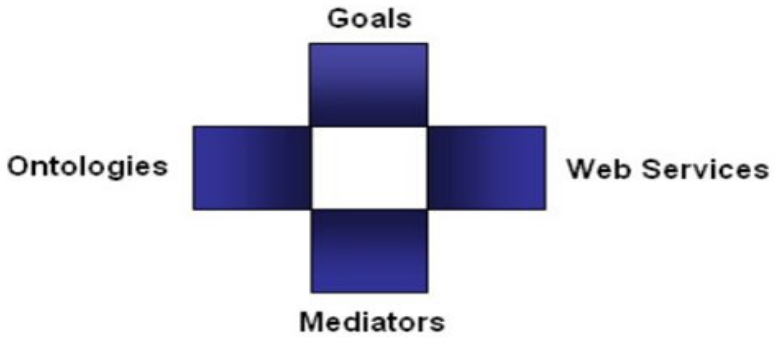


Figure 2.5 : Composants de WSMO.

- ***Ontologies :*** elles représentent un élément clé dans le WSMO parce qu'ils fournissent des terminologies pour décrire les autres éléments. Son objectif est de définir de la sémantique formelle de l'information et de relier la machine et la terminologie de l'homme.

- ***Web Services :*** ils représentent un élément atomique des fonctionnalités qui peuvent être réutilisées pour en construire de plus complexes. Afin de permettre leur découverte, l'invocation de composition, d'exécution, de suivi, de médiation et d'indemnisation, les Web services sont décrits dans WSMO à partir de trois points de vue différents: les propriétés non fonctionnelles, les fonctionnalités et le comportement. Ainsi, un Web service est défini par *ses propriétés non fonctionnelles*, *ses ontologies importées*, *ses médiateurs utilisés*, *sa capacité* et *ses interfaces*.
- ***Objectifs:*** ils décrivent les aspects liés aux désirs des utilisateurs à l'égard de la fonctionnalité demandée par opposition à la fonctionnalité fournie décrite dans la capacité du Web service.
- ***Médiateurs :*** ils décrivent les éléments qui visent à surmonter l'inadéquation structurelle, sémantique ou conceptuelle qui apparaissent entre les différentes composantes qui construisent une description WSMO. Actuellement, le cahier des charges porte sur quatre différents types de médiateurs:
  - ***OOMediators*** : L'importation de l'ontologie cible dans l'ontologie source pour résoudre tous les décalages de représentation entre la source et la cible;
  - ***GGMediators*** : la connexion des buts retrouvés dans une relation de raffinement et résolution des inadéquations existants entre eux;
  - ***WGMediators*** : Lien de Web services à des objectifs et résolution des inadéquations;
  - ***WWMediators*** : Connexion de plusieurs Web services pour la collaboration.

5.3 METEOR-S

METEOR (Managing End-To-End OpeRations) est un projet réalisé dans le laboratoire LSDIS dans l'université Georgia. METEOR-S (METEOR for Semantic Web Services) a pour objectif de fournir des Web services sémantiques dans le cadre du projet METEOR **[METEOR-S]**. Il propose un système basé sur l'annotation sémantique de descriptions

WSDL et fournit un canevas représentatif pour intégrer la sémantique des données, la sémantique fonctionnelle, la qualité de service et l'exécution de Web services.

METEOR-S permet de décrire une approche automatique pour l'annotation sémantique de la description WSDL d'un Web service. Il définit les ontologies basées sur le langage OWL pour représenter des services et permet de définir une composition de services par un processus abstrait qui spécifie le flot d'opérations. METEOR-S a choisi BPEL4WS comme langage de spécification de ces processus abstraits.

Pour accomplir ces tâches, METEOR-S définit quatre niveaux de sémantiques : de données, fonctionnelle, de qualité de service et d'exécution.

- **Sémantique des données** : est l'annotation (description sémantique) des entrées/sorties et des messages d'erreurs d'un Web service. Dans METEOR-S cette sémantique est représentée en utilisant l'ontologie RosettaNet.
- **Sémantique fonctionnelle** d'un Web service est décrite via les opérations offertes par un Web service. Chaque opération est décrite par l'ensemble des données échangées par l'opération, la fonctionnalité de l'opération ainsi que ses pré-conditions et post-conditions.
- **Sémantique de qualité de service** caractérise des aspects tels que la performance d'un Web service. Cette spécification doit être comprise par tous afin que les clients puissent choisir au mieux les services qui répondent à certaines de leurs exigences.
- **Sémantique d'exécution** qui permettra d'effectuer des tests de validation du service.

Le projet METEOR-S s'est développé selon trois phases principales qui ont permis d'introduire les trois concepts importants de cette initiative **[IZZ06]** :

- MWSDI (*METEOR-S Web Service Discovery Infrastructure*) qui consiste à installer une infrastructure de découverte sémantique définie au dessus d'un registre UDDI **[VER+05]**.
- MWSAF (METEOR-S Web Service Annotation Framework) qui permet d'enrichir sémantiquement les Web services en utilisant une extension améliorée de WSDL appelée WSDL-S (WSDL Semantics) **[WSDL-S]**. Le rôle de ce dernier est d'enrichir sémantiquement les fichiers WSDL et également le registre UDDI.
- MWSCF (METEOR-S Web Service Composition Framework) pour la composition et l'exécution des services pour lesquelles METEOR-S a choisi BPEL4WS.

PARTIE 2 : Découverte des Web services

6. Problématique de la découverte des Web services

L'objectif principal de la technologie des Web services est de fournir l'intégration et l'interaction dynamique des systèmes hétérogènes, et de ce fait, faciliter la coopération rapide et efficace entre les différentes entités dans un environnement collaboratif **[EME+04]**, **[SAH+05]**, **[ESS+05]**.

Pour atteindre ces objectifs, les Web services doivent agir automatiquement avec le minimum d'intervention humaine; ils doivent être capables de découvrir d'autres services qui ont des capacités particulières et réalisent des tâches précises **[ESS+05]**.

A cause de ces contraintes, la découverte des Web services est conçue comme un problème critique depuis la naissance de ce paradigme. La première génération des travaux de recherche effectuée sur les architectures des Web services, proposent des solutions basées sur les méthodes de découverte centralisées (comme UDDI) où les Web services sont décrits par les interfaces (signatures) de leurs fonctions. Dans une telle architecture, les fournisseurs des Web services publient les capacités et les fonctionnalités des ces services dans un enregistrement central.

Cependant, ces méthodes souffrent d'un nombre de problèmes classiques connus dans les systèmes centralisés comme le seul point centralisé (point d'échec) et le coût élevé de la maintenance. De plus, ces méthodes ne garantissent pas l'évolutivité (scalabilité) requise pour soutenir un environnement flexible et dynamique **[MAN+07]**, **[HU+05]**.

Les paragraphes suivants présentent trois types de solutions centralisées ainsi que les inconvénients et les problèmes rencontrés au niveau de chaque solution.

6.1 Méthodes centralisées de découverte

Les méthodes de découverte centralisées présentent la première génération des travaux proposés dans ce domaine. La première initiative industrielle et académique issue pour la découverte des Web services est la recommandation UDDI. Ce dernier a été utilisé de deux façons différentes, tant qu'un annuaire universel ou privé.

6.1.1 Annuaire universel

La première version de la spécification UDDI a été publiée en septembre 2000 par Ariba, IBM et Microsoft **[CHA02]**. La philosophie de UDDI dans cette version est de centraliser les

points d'entrées des Web services, c'est à dire la création d'un annuaire universel pour les Web services fournis par les différents propriétaires (entreprises ou individus). Cet annuaire universel recueillant les inscriptions des fournisseurs de Web services permet à un chacun d'effectuer des recherches pour trouver les services particuliers dont il a besoin **[CHA02]**.

Un exemple de ce type d'annuaire est celui qui regroupe les annuaires UDDI des opérateurs Microsoft, IBM et XMethods (il été accessible à partir de l'adresse suivante : *http://soapclient.com/index.html)*. Ce site fournit aux utilisateurs la possibilité de chercher des services Web en utilisant des différents critères : Business Names (Business Entity), Service Names (Business Service), Service Types (tModel) et autres **[GHA04]**.

Cette proposition fut l'objet d'un certain nombre de et pose deux grands problèmes **[MOD02]**:

- Le manque de modération dans les enregistrements UDDI existants : les modérateurs sont responsables de régler les teneurs des enregistrements qu'ils modèrent (des compagnies posant comme fournisseurs des services qu'ils ne présentent pas).
- L'absence de sécurité et de fiabilité pour les échanges B2B particulièrement dans le cas de paiement électronique.

A cause de ces problèmes, cet annuaire universel a été supprimé par ses opérateurs en 2006. D'autres solutions ont été développées pour le remplacer. Dans ce qui suit, nous présentons les annuaires privés et les moteurs de recherche.

6.1.2 Annuaires privés

Comme UDDI a obtenu le soutien des entreprises et la plupart des fournisseurs des Web services, son utilisation n'était pas limitée aux échanges B2B, mais aussi dans le domaine du commerce électronique (B2C : Business to customers). C'est dans ce contexte que la deuxième version d'UDDI a été publiée en juin 2001 ayant pour but de corriger quelques problèmes apparaissants dans la première version. Cette version est moins lourde à mettre en œuvre, elle est en plus adaptée aux échanges B2B *privés* (sur un Extranet), au commerce électronique (dans un contexte B2C) et même à l'intégration des applications à l'intérieur de l'entreprise.

Dans le contexte B2C, une entité commerciale implémente un ensemble de registres UDDI privés ou semi- privés. Généralement, l'entité commerciale a certaines règles commerciales ou des intérêts à suivre pour personnaliser les résultats de la découverte des Web services. Par

exemple, une entité commerciale, qui possède un UDDI privé, veut toujours contrôler l'affichage des résultats correspondants aux requêtes des consommateurs **[TAN+06]**.

Néanmoins pour les autres initiatives qui ont vu le jour, comme WSIL (Web Services Inspection Language) **[WSIL]** et ebXML (e-business eXtensible Markup Language) **[ebXML]**, UDDI reste la spécification la plus utilisée par les industriels, et le standard le plus cité dans les travaux de recherche. Cependant, même dans sa deuxième version, UDDI est incapable de supporter les environnements dynamiques et flexibles tout particulièrement dans le cas de la composition dynamique des Web services. En plus, UDDI offre seulement un contenu technique à cause de la publication des Web services à l'aide de la description WSDL.

6.1.3 Moteurs de recherches

A cause de la description technique des Web services, il n'est pas évident de retrouver ces derniers en utilisant un moteur de recherche publique comme Google. Un Web service est un composant logiciel encapsulant des fonctionnalités métiers. Il est considéré comme une boite noire (on ne connaît pas en détail son implémentation) et n'expose que des données techniques décrites par son interface WSDL.

Dans la littérature, il n'existe qu'une seule initiative qui offre un moteur de recherche spécifique dédié à la découverte des Web services, c'est « ***service finder*** » **[finder]**. Les initiateurs de ***service finder*** visent à développer une plate-forme de découverte des services dans laquelle les Web services sont intégrés dans un environnement Web 2.0.

Cependant, dans sa première version béta lancée en 2008 (actuellement disponible à http://demo.service-finder.eu/), ce moteur de recherche pose presque les même problèmes déjà reconnus dans l'UDDI universel. Pour le moment il ne supporte pas la découverte automatique (en temps d'exécution) effectuée par des machines. Néanmoins, ***service finder*** a connu des améliorations sur le plan sémantique par rapport à l'UDDI universel en utilisant quelques techniques du Web 2.0.

6.2 Discussions

Les méthodes centralisées présentées dans la section précédente souffrent de trois problèmes principaux : la prise en compte de la sémantique pour la découverte des Web services, le non considération de la Qualité de Service (QoS) et la composition dynamique des Web services **[TAM06]**.

a) Problème de la sémantique

WSDL et UDDI ne permettent pas à des Web services d'interagir de manière intelligente. Ils ne décrivent pas suffisamment les connaissances de manière à les rendre exploitables par des machines. Dans ce contexte, la sémantique est considérée comme un élément primordial qui permet de représenter et de manipuler des connaissances de natures différentes : structurelles, fonctionnelles et même opérationnelles. De ce fait, en plus de la recherche par mots-clés, la découverte de services se fera selon les différents concepts qui leurs sont associés et le contexte dans lequel ils doivent être compris. Dans une telle situation, la découverte des Web services est caractérisée par des capacités de raisonnement qui s'appuient sur une description formalisée et sur la mise en relation des différentes sources d'information **[TAM06]**.

b) Problème de qualité de service (OoS)

Les Web services proposés sur Internet par différents fournisseurs sont nombreux. Dans une telle situation, il n'est pas évident de découvrir un service Web qui réponde à un besoin d'un demandeur. Pour effectuer le bon choix entre les différents Web services candidats, diverses propriétés de qualité de service telles que la disponibilité, la performance et la fiabilité sont nécessaires. Généralement, les fournisseurs publient des descriptions concernant les propriétés fonctionnelles des services qu'ils proposent. Par contre, des informations, comme le temps de réponse et la disponibilité des services, ne sont jamais fournies par ces fournisseurs. Par exemple, dans un environnement des échanges entre partenaires (B2B), il est important d'en trouver le meilleur. Il devient donc nécessaire de pouvoir extraire et exploiter les Web services pertinents parmi ceux qui ont été collectés **[TAM06]**.

c) Problème de composition dynamique des Web services

Comme nous l'avons déjà mentionné, il existe deux types de méthodes pour la composition des Web services : orchestration et chorographie. Dans ces deux catégories, nous pouvons distinguer deux techniques permettant la mise en place d'une composition : statique et dynamique. Dans une composition statique, les services candidats ainsi que les messages échangés entre ces services sont connus préalablement, c'est à dire, au temps de la construction (Buildtime). Dans ce cas, la composition s'exécute de la même manière pour différentes instances ; alors que dans une composition dynamique les services impliqués sont découverts au temps de l'exécution (Runtime). Ce type de compositions n'est pas exécuté de manière prévisible et répétitive. Il est donc évident que la découverte des Web

services, dans une composition dynamique, est une tâche délicate qui nécessite la capacité de sélectionner, de composer et de faire interagir des services existants dynamiquement selon les besoins.

Dans les méthodes centralisées, ce problème est plus raisonnable parce qu'UDDI ne propose pas de solutions pour les compositions dynamiques de services. Il n'a pas la capacité d'assembler et d'orchestrer des Web services à cause de la description de ceux proposés par UDDI ; cette dernière n'intègre que très peu d'aspects sémantiques.

7. Découverte distribuée des Web services

D'autres méthodes de découverte et de composition des Web services ont été proposées pour résoudre les limites des méthodes centralisées. Ce type de méthodes est caractérisé par la distribution du mécanisme de publication et de découverte. Elles ont pour objectif principal la composition dynamique des Web services et il existe deux techniques de bases qui sont utilisées pour implémenter les méthodes décentralisées : les Systèmes Multi Agents et les réseaux P2P.

7.1 Utilisation des SMA (Systèmes Multi Agents)

Le couplage des Web services et des systèmes multi agents est bidirectionnel. D'une part les agents peuvent utiliser les Web services pour exposer leurs capacités au monde extérieur. Dans ce cas, les SMA hétérogènes utilisent les Web services pour raison d'interopérabilité **[SEG+04]**. D'autre part, les SMA sont utilisés généralement pour implémenter la composition chorographique des Web services.

Dans ce type d'architecture, chaque agent est responsable d'un ou de plusieurs Web services. Sont but est la négociation et la communication d'une manière intelligente avec les autres agents, afin d'accomplir un objectif commun à travers la composition de plusieurs services **[VAD+11]**, **[BOU+07]**, ces derniers pouvant appartenir à différents fournisseurs.

Cependant, les SMA sont généralement utilisés pour composer les Web services dans un environnement coopératif fermé (pour échanges B2B par exemple) **[BRA+09]**. Habituellement, ce type de solutions ne propose pas un mécanisme de découverte des Web services. L'objectif principal des ces solutions est la proposition d'une architecture SMA ainsi que des protocoles et des algorithmes pour composer dynamiquement les Web services.

7.2 Utilisation des réseaux P2P

Le deuxième type des méthodes décentralisées dédiées à la découverte et la composition des Web services permet d'implémenter les systèmes P2P. Ces derniers ont été impliqués dans ce contexte parce qu'ils fournissent une alternative évolutive aux systèmes centralisés et ce en distribuant les Web services sur tous les pairs du réseau. De plus, la convergence entre les technologies du Web sémantique, des Web services et du P2P, présente des nombreux avantages, notamment pour le partage des connaissances dans les réseaux à grande échelle. Dans ce contexte, nous allons présenter dans ce qui suit quelques travaux de recherche proposés dans ce contexte. D'autres travaux seront décrits dans le chapitre 4.

WSPDS (Web Services Peer-to-peer Discovery Service) est une architecture de découverte des Web services dans un réseau P2P pur (non structuré) **[BAN+04]**. Elle est composée essentiellement de deux modules : un moteur de communication et un moteur de traitement des requêtes. De plus, elle offre une interface pour les utilisateurs externes (simples internautes ou autres applications).

Le fonctionnement des WSPDS est simple : les pairs, qui implémentent cette architecture, forment un espace collaboratif pour la découverte des Web services. Lorsqu'il reçoit une requête, un pair participant utilise le moteur des requêtes pour rechercher un Web service qui répond à cette requête, il envoi la réponse (si elle existe) ou il transmet la requête aux pairs voisins et renvoi les réponses au pair demandeur. La réception et la propagation des requêtes s'effectuent en utilisant le moteur de communication.

Pour chaque Web service, une description WSDL, annotée sémantiquement, est disponible, en plus d'un profile DAML-S qui est basé sur une ontologie DAML+OIL qui décrit les fonctionnalités du Web service. De plus, WSDPS utilise des fichiers WSIL pour fournir des métadonnées additionnelles (nom du service, petite description) et pour associer le Web service au document WSDL annoté correspondant. Cependant, et malgré l'utilisation du même langage de description (DAML-S), WSDPS présente le problème de l'hétérogénéité sémantique au niveau des concepts utilisés : chaque pair participant utilise ses propres concepts pour décrire les Web services fournis. A ce souci se rajoute le fait que WSPDS soit exclusivement conçu pour la découverte des Web services basiques ; il ne permet pas la composition des Web services.

Une deuxième approche qui exploite les P2P non structurés et DAML-S est celle proposée par M. Paolucci et al. **[PAO+03]**. Dans cette approche, tous les pairs, d'un réseau Gnutella,

présentent la même architecture organisée en trois couches : un parseur DAML, un processeur DAML-S et une couche de communication avec les autres pairs du réseau. Le parseur DAML converti les spécifications DAML-S en un ensemble de règles exploitables par le processeur DAML-S. Le but de ce dernier est d'effectuer le mapping (Machmaking) et interagir avec le monde extérieur à travers la couche de communication.

Malgré l'uniformisation des spécifications DAML-S proposée dans cette approche par apport à l'architecture WSPDS, le travail de M. Paolucci et al. **[PAO+03]** pose le même problème de l'hétérogénéité sémantique et se révèle insuffisant pour décrire le mécanisme de composition des Web services découverts.

Dans le travail proposé à **[VER+05]**, K. Verma et al. présentent l'architecture METEOR-S WSDI (METEOR-S Web Service Discovery Infrastructure ou MWSDI) qui est un système distribué basé sur la typologie hybride. MWSDI est constitué d'un ensemble de Super-pairs contenant ses propres registres, et collaborant entre eux pour répondre aux requêtes des demandeurs. Une ontologie, d'un (ou plusieurs) domaine (s), est associée à chaque registre. Ce dernier est implémenté par un annuaire UDDI qui supporte la découverte et la publication sémantiques des Web services (en utilisant une annotation sémantique).

MWSDI présente quatre types des pairs selon leurs rôles dans le réseau :

- Pairs opérateurs : ce sont les Super-pairs qui implémentent les registres UDDI,
- Pairs intermédiaires : ce sont les pairs effectuant l'enregistrement des ontologies des registres lorsque de nouveaux Super-pairs rejoignent le réseau avec leurs propres registres,
- Pairs auxiliaires : sont des pairs qui fournissent des registres mais qui ne sont pas considérés comme des pairs opérateurs. Cependant, ils peuvent être des pairs opérateurs à tout moment.
- Pairs clients : ce sont des pairs simples qui peuvent seulement publier ou rechercher des Web services auprès des pairs opérateurs.

Comme nous avons déjà mentionné dans la section 5.3, MWSDI présente une partie du projet METEOR-S qui se compose de deux autres composants : MWSAF et MWSCF conçues pour la gestion de la sémantique et la composition des Web services respectivement. Pour cette raison, MWSDI présente un problème de lourdeur concernant son fonctionnement et par conséquent, il est moins adapté aux grands réseaux P2P et aux environnements caractérisés par une dynamique élevée.

Une autre architecture, qui implémente la typologie hybride, est celle de GloServ (**Glo**bal **Serv**ice discovery architecture) **[ARA+07]**. Cette solution est néanmoins critiquée car les pairs reliés à un Super-pair sont organisés selon le protocole P2P structuré CAN (Content Addressable Network) **[RAT+01]** et les pairs sont catégorisés selon les concepts ontologiques décrivant les services fournis. Pour gérer l'infrastructure sémantique, GloServ utilise le langage OWL-DL où chaque concept est décrit par une classe OWL avec un ensemble propriétés.

Cependant, GloServ ne décrit pas suffisamment comment un pair peut joindre le réseau surtout dans le cas où un pair peut fournir des services de différents domaines avec différents concepts. De plus, GloServ ne définit pas un mécanisme de composition des Web services.

Le tableau suivant résume les descriptions des différents travaux présentés dans cette section.

Systèmes	Typologie P2P	Protocole	Langage sémantique
WSDPS	Non structurés	Gnutella	DAML-S/ DAML+OIl
[PAO+03]	Non structurés	Gnutella	DAML-S
METEOR-S WSDI	Hybride	/	DAML-S
GloServ	Hybride/ structurés	CAN	OWL-DL

Tableau 2.2 : Architectures basées P2P pour la découverte des Web services sémantiques.

D'autres travaux de recherches, s'inscrivant dans ce contexte, sont bien décrits dans **[BIA+06]** avec une étude comparative détaillée.

8. Conclusion

Aujourd'hui, la faiblesse des progiciels réside principalement dans la dynamique des systèmes d'information. L'architecture orientée services est une nouvelle vision qui répond d'une manière efficace aux problèmes des systèmes d'information en terme de réutilisabilité et d'interopérabilité.

Une solution SOA est habituellement basée sur les services qui sont habituellement employés pour agir l'un avec l'autre. Le système d'information est alors vu comme une collections de services qu'il est possible de composer pour créer des services avec un plus

haut niveau de granularité et permettant d'accéder de manière transparente aux applications du système d'information.

Dans ce chapitre, nous avons présenté l'architecture SOA de manière abstraite, pour ensuite discuter sa concrétisation en utilisant les Web services. Nous avons exposé les différentes technologies adjacentes aux Web services en insistant tout particulièrement sur la composition des Web services comme moyen d'implémentation des processus métiers. A ce stade, nous avons présenté le langage le plus approprié pour composer des Web services à savoir le BPEL.

Nous avons ensuite présenté les limites des Web services et de leurs compositions en termes de découverte des Web services en particulier au niveau de leur interopérabilité sémantique. Dans ce cadre, nous avons étudié l'apport de la technologie du Web sémantique aux Web services et avons expliqué le concept de Web services sémantiques ainsi que les différents langages et frameworks proposés dans ce domaine. De plus, nous avons présenté les problèmes rencontrés au niveau des méthodes classiques dédiées à la découverte et la composition des Web services. Nous avons aussi présenté quelques travaux de recherches qui décrivent des architectures distribuées pour la découverte des Web services sémantiques.

Dans le chapitre suivant, nous allons décrire notre approche qui utilise des technologies du Web sémantique et celles des systèmes P2P pour offrir une solution distribuée permettant la découverte et la composition dynamiques des Web services sémantiques.

Chapitre 3

Une stratégie de découverte et de composition des Web services

1. Introduction

Dans le chapitre précédent, nous avons mentionné que les services sont conçus pour être composés en services plus complexes. La composition des Web services est une opération qui se déroule différemment d'un scénario à un autre bien qu'elle soit toujours fortement liée à une étape préalable qui est la découverte des Web services. Cette dernière présente une phase primordiale pour collecter les services candidats à une composition tout en assurant leur qualité, leur compatibilité et la faisabilité de la composition.

Nous avons aussi expliqué le bénéfice d'utilisation le Web sémantique pour décrire les différents Web services fournis. Les langages de description sémantique issus (OWL-S, WSMO,…) permettent de décrire les connaissances liées au domaine d'application d'un Web service, ce qui rend sa manipulation plus intelligible et exploitable par la machine. Cette caractéristique offre un atout pour la découverte automatique des Web services et implicitement pour sa composition dynamique.

Après que le Web sémantique ait résolu le problème des standard de descriptions classiques comme WSDL et UDDI, qui ne décrivent pas suffisamment les connaissances d'un Web service, il s'est avéré que la composition dynamique des Web services nécessite un moyen plus efficace qu'UDDI qui ne permet pas d'agréger des Web services d'une manière adaptative et correcte.

Afin de régler ce problème, les systèmes P2P ont été impliqués dans ce domaine. Le paradigme P2P est considéré comme une évolution pour les systèmes distribués qui se concentre sur la gestion des réseaux et le partage des ressources avec une meilleure fiabilité et scalabilité. Les systèmes P2P ont été inclus dans plusieurs domaines de recherche comme le travail collaboratif et la découverte des ressources pertinentes y compris les Web services. Les systèmes P2P fournissent une alternative évolutive aux systèmes centralisés en distribuant les Web services sur tous les pairs du réseau.

Le travail présenté dans ce chapitre consiste principalement en une proposition d'une approche permettant la découverte et la composition dynamique des Web services sémantiques dans les systèmes P2P.

En effet, après avoir décrit la problématique des méthodes de découverte centralisées, nous allons présenter une stratégie générique pour la découverte distribuée des Web services sémantiques. Dans cette stratégie, nous employons les systèmes P2P non structurés pour proposer un algorithme épidémique permettant la recherche des Web services. L'approche

préconisée implémente une table distribuée, dite « table de composition », pour préserver la trace des Web services déjà composés dans le réseau. Cette table vise à accélérer le temps de recherche des Web services à travers la réutilisation des compositions déjà réalisées. De plus, nous allons montrer les avantages de cette table toute en assurant la cohérence de son contenu à travers des algorithmes de notification.

2. Une stratégie de découverte et de composition des Web services sémantiques dans les réseaux P2P non structurés

Dans cette section, nous allons d'abord motiver l'utilisation des réseaux P2P non structurés dans notre solution. Ensuite, nous présentons les deux idées de base de cette solution qui sont l'algorithme épidémique dédié à la découverte et la composition des Web services sémantiques, et la table de composition.

2.1 Motivation d'utilisation des réseaux P2P non structurés

Dans notre travail, nous nous intéressons aux systèmes décentralisés, spécialement aux réseaux P2P non structurés. Comme présenté dans le chapitre 1, il existe deux grandes classes des systèmes décentralisés: structurés et non structurés. Les systèmes structurés utilisent généralement une table de hachage (DHT) pour envoyer des requêtes et pour localiser les ressources. Les réseaux structurés ont été proposés pour résoudre un nombre important de problèmes connus au niveau des réseaux non structurés et hybrides. Ils ont plusieurs avantages comme : l'évolutivité, la disponibilité et la gestion de la tolérance aux pannes. Cependant, les protocoles à base de DHT sont difficiles à mettre en place à cause de leurs typologies statiques (par exemple, un anneau pour Chord) **[SAR+04]**. De plus, très peu de d'implémentations sont proposées pour ce type de réseaux. La plupart des tentatives restent beaucoup plus académiques (dans les laboratoires comme Open Chord **[OpenChord]**), elles ne sont pas utilisées par la communauté des utilisateurs P2P. Enfin, la DHT ne permet pas d'effectuer des recherches complexes ou des recherches par contenu.

D'autre part, les réseaux P2P non structurés (comme Gnutella) ne passent pas à l'échelle supérieur parce qu'ils utilisent des algorithmes de recherche basés sur les inondations. Une requête génère une grande quantité de trafic où les grands systèmes deviennent rapidement surchargés **[LV+02]** bien que ce type de réseaux ne nécessite ni des annuaires centralisés ni un contrôle précis sur la topologie du réseau ou le placement des données (ou des ressources).

La décision d'utiliser les réseaux P2P non structurés dans ce travail est justifiée par plusieurs raisons. Généralement, dans le contexte de la découverte des Web services sémantiques, le demandeur (le service consommateur) recherche une capacité, un but ou une propriété d'une ressource (Web service). Alors, quand le demandeur envoie une requête, il n'est pas nécessaire de connaître au préalable l'emplacement ou l'identificateur de la ressource. De ce fait, les réseaux non structurés sont conçus pour implémenter une recherche par contenu en utilisant généralement un (ou plusieurs) mot clé. Alors que, ces caractéristiques ne sont pas fournies par les systèmes P2P structurés où pour retrouver une ressource il est nécessaire que le demandeur connaisse à l'avance l'identifiant de la ressource. De plus, les systèmes P2P non structurés sont utilisés par de très grandes communautés d'internautes **[SAR+04]**. Ainsi, les systèmes P2P non structurés sont plus adaptés à la découverte des services distribués.

Dès lors, dans notre travail, nous allons proposer une stratégie de découverte et de composition des Web services dans les systèmes P2P non structurés. En effet, pour simplifier le phénomène de passage à l'échelle dans ce type de réseaux, nous proposons une stratégie de réplication qui résout ce problème tout en réduisant le trafic dans le réseau et accélérant la découverte des services. Pour atteindre cet objectif, nous proposons l'utilisation d'une table de réplication (table de composition) qui préserve les traces des Web services déjà composés. Cette table présente de nombreux avantages, qui seront présentés dans la section suivante.

3. Description générale de la stratégie

Rappelons que l'objectif principal de cette solution est de fournir un scénario purement décentralisé qui supporte la composition automatique des Web services sémantiques distribués sur un réseau P2P. A partir d'un réseau purement non structuré, nous voulons composer un Web service qui répond à une requête particuliaire. Ce service peut être composé de plusieurs services appartenant à différents pairs. L'objectif est de créer un espace collaboratif entre les différents pairs participants à la réalisation d'un but commun.

Dans cette première étape de notre travail, nous décrivons une stratégie de découverte et de composition des Web services dans un réseau P2P non structuré. Cette phase consiste à fournir un algorithme distribué pour la découverte et la composition des Web services sémantiques. L'idée principale est de présenter un algorithme épidémique qui permet de rechercher des services distribués sur tous les pairs du réseau afin de composer de nouveaux services personnalisés. Aussi, dans cette solution, nous avons utilisé une table –dite table de

composition- (table 3.1) afin de préserver la trace du chemin de composition pour une éventuelle future réutilisation. Cette table est considérée comme une mémoire cache utilisée pour accélérer la recherche des Web services déjà composés **[GHA+09]**.

3.1 Algorithmes épidémiques basés sur l'input/output Matchmaking

Pour découvrir les Web services appropriés, nous avons proposé un algorithme épidémique (par inondation) basé sur les entrées/sorties et orienté par le but à réaliser. Dans cette solution, tous les pairs participants sauvegardent, dans leurs tables de composition, toutes les traces des Web services déjà composés dans le réseau. Cet historique peut être utilisé pour donner des réponses rapides pour d'éventuelles requêtes similaires à celles traitées précédemment.

Avant de présenter l'algorithme proposé et la table de composition, il faut d'abord définir les concepts suivants **[GHA+09]** :

Définition 1 : Un Web service est 3-uplet (Entrée, Sortie, But)

Définition 2 : Une composition est un processus constitué d'un ensemble de Web services. Dans notre contexte, nous avons deux types de compositions : composition-locale et composition-P2P.

Définition 3 : Un Web service composé localement est construit à partir d'un ensemble de Web services d'un même pair.

Définition 4: Un pair participant est un pair qui collabore dans une composition-P2P.

Définition 5 : Une composition-P2P est une composition d'un service à partir d'un ensemble de Web services qui appartiennent à différents pairs du réseau. Une composition-P2P est un 5-uplet (Pair-Initiateur, ID-Comp, Init-input, Init-Output, But) où :

- **Pair-Initiateur** : Le premier pair dans l'enchainement (qui exécute le (les) premier Web service) est appelé le pair initiateur (le pair initiateur est un pair participant).
- **ID-Compo** : l'identificateur de composition. Chaque composition a un seul identifiant composé de deux valeurs (Pair-Initiateur, N°-composition). Lorsque deux compositions ont le même pair initiateur, elles ont automatiquement deux numéros différents, parce que le numéro de composition est généré par le pair initiateur lors de la sauvegarde de la composition.
- **Init-Input** : l'entrée initiale du Web service composite.
- **Init-Output** : la sortie initiale du Web service composite.

- **But** : le but du Web service. Un but est une conceptualisation du domaine des services dont les objectifs finaux sont identiques ou similaires. Le but d'un Web service est spécifié dans sa description sémantique **[MAN+07]**.

3.1.1 Algorithme principal

L'algorithme principal de la découverte est un algorithme épidémique qui assure la progression de l'opération de la rechereche dans le réseau. Chaque pair execute cet algorithme lorsque il reçoit une requête de découverte d'un Web service. Il est définit comme suit:

Début

1. Le pair reçoit une requête de type « search (init-Input, Init-Output, But) ».
2. Chercher un Web service (WS) local qui répond à la requête.
3. **Si** (il existe un WS local) **alors**; envoyer la réponse favorable; Aller à **Fin**;
4. **Si** (il n'existe pas un WS local) **alors**
   Essayer de composer un Ws localement pour répondre à la requête.
5. **Si** (c'est possible de composer un WS localement) **alors** envoyer la réponse favorable; aller à **Fin**;
6. **Si** (il n'existe pas une composition locale qui répondre à la requête) **alors Search-in-Composition-table (Init-Input, Init-Out-put, But);**
7. **Si (**il n'existe pas une composition dans la table) **alors Launch-a-New-P2P-composition ( );**
8. **Fin.**

L'opération de recherche dans la table de composition présentée dans l'étape 6 de l'algorithme précédent (**Search-in-Composition-table (Init-Input, Init-Out-put, But)** est implémentée par l'algorithme suivant :

Début

Select [Pair-Initiateur, ID-Compo] **from** la table de composition **where** [(l'entrée initiale de la composition=Init-input) **&** (la sortie initiale de la composition=Init-Output) **&** (le but de la composition=But) **&** (l'état de la composition=Actif)]; /*le résultat de cette sélection est une liste nommée selection-list*/

1. **SI** (selection-list est non vide) **alors**
   Tant que (selection-list est non vide) **Faire**
   a) Sélectionner un Pair-Initiateur de la liste selection-list;
   b) Envoyer un message: **search** (ID-Compo, Init-Input, init-Output, But) au Pair-Initiateur;
   c) **SI** (la découverte est terminée avec succès) **alors** envoyer la réponse; Aller à Fin; /* La Fin du programme principale*/

Fin Tant que;

Fin.

Le phénomène de l'épidémie émerge parce que l'algorithme **Launch-a-New-P2P-composition();** est exécuté de la même manière par tous les pairs participants dans l'opération de découverte, sauf que le paramètre en entrée ou en sortie est modifié selon le type d'enchainement (en avant ou en arrière).

Pour lancer une nouvelle opération de composition dans le réseau (présentée par la fonction **Launch-a-New-P2P-composition ( );** dans l'étape 7 de l'algorithme principal), nous avons proposé deux algorithmes de découverte basés sur la logique de planification : chainage en avant et chainage en arrière.

3.1.2 Algorithme du chainage avant

Dans le chainage avant, chaque pair cherche un Web service qui a une entrée et qui peut faire progresser le processus de découverte. Cet algorithme est présenté comme suit :

Début

1. Créer une liste des Web services qui ont une entrée= Init-Input (List-WS-init-Input).
2. **Tant que** (List-WS-Init-Input non vide) **Faire**
   a) Sélectionner l'entête de la liste List-WS-Init-Input;
   b) Init-Input:= la sortie du Web service; calculer le TTL;
   c) envoyer un message (Pair-Initiateur, Iinit-Input, Init-Output, But, TTL) au pairs voisins;
   d) **SI** (temps ≤ TTL & une réponse favorable) **alors**
   Recherche terminée; envoyer la réponse; Aller à Fin;
   Sinon
   SI (temps ≤ TTL & réponse défavorable) **alors**
   Sélectionner le service suivant dans la liste list-WS-Init-Input;
   Fin Tant que; Fin.

3.1.3 Algorithme du chainage arrière

Dans le chainage arrière, chaque pair cherche un Web service qui dispose d'une sortie qui peut faire progresser le processus de découverte. Cet algorithme est présenté comme suit :

```
Début
1. Créer une liste des Web services qui ont une sortie= Init-Output
(List-WS-init-Output).
2. Tant que (List-WS-Init-Output non vide) Faire
  a) Sélectionner l'entête de la liste List-WS-Init-Output;
  b) Init-Output := l'entrée du Web service; calculer le TTL;
       c) envoyer un message (Pair-Initiateur, Iinit-Input, Init-
     Output, But, TTL);
       d) SI (time ≤ TTL & réponse favorable) alors
       Recherche terminée; envoyer la réponse; Aller à Fin;
           Sinon
         Si (temps ≤ TTL & réponse défavorable) alors
     Sélectionner le service suivant dans la liste list-WS-Init-
  Output;
    Fin Tant que;
  Fin.
```

La figure 3.1 présente les deux types de chainages (le pair P1 est le pair initiateur).

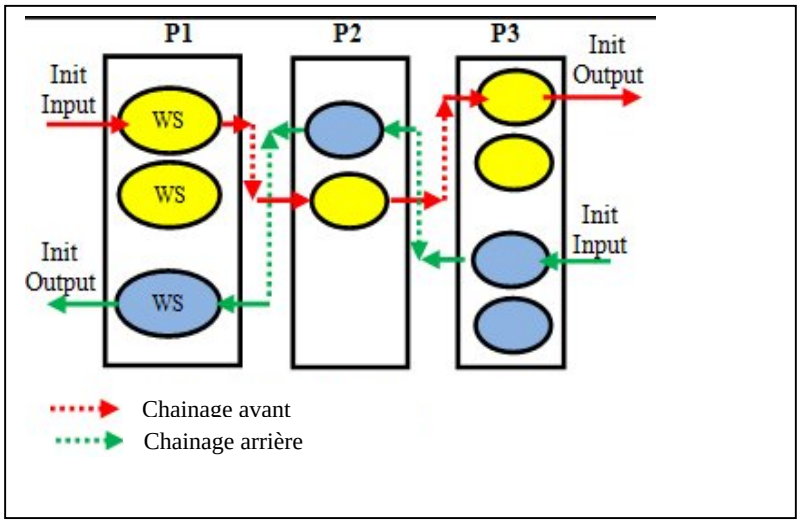


Figure 3.1: Algorithmes en chaînage avant et en chaînage arrière.

3.2 Contenu de la table de composition

Dans la stratégie proposée, nous avons utilisé une table -dite table de composition- distribuée sur tous les pairs participants du réseau. Elle est constituée d'un ensemble d'attributs qui décrivent les différentes compositions réalisées dans le réseau (voir table 3.1).

Pair-Initiateur	ID-Compo	Init-input	Init-output	But	Services exécutés	Services réserves	Pairs Précédents	Pairs suivants	Pairs réserves	Etat de la compo
P11	0001	….	…..	… …	……..	……	…….	……	……..	……
P2	0003	……..	…… …..	… … ….	……..	……..	P4	P7	…….	…….
….	……	…….	……	… … …	……...	……..	……..	…….	…….	…….
P23	0010	…….	……..	… ….	……..	…….	……..	…….	……..	…….

Tableau 3.1 : Table de composition.

Chaque pair du réseau obtient une table de composition qui contient des données sur les différentes compositions où il a déjà participé à leur réalisation. Une composition est donc enregistrée dans plusieurs tables réparties sur plusieurs pairs. Pour chaque composition, les attributs suivants doivent être identiques dans toutes les tables : le pair initiateur, l'entrée initiale (Init-Input), la sortie initiale (Init-Output), le but et l'état de la composition. Les autres attributs présentent des données locales pour chaque pair participant (services exécutés, services en réserves, pairs précédents, pairs suivants et pairs en réserves). Ces attributs sont définis comme suit :

Services exécutés : les Web services exécutés localement par un pair donné pour composer un Web service dans le réseau.

Services en réserves: un Web service qui peut remplacer un service exécuté localement dans une composition-P2P.

Pairs Précédents : les pairs qui exécutent les Web services précédents dans une composition-P2P.

Pairs suivants : les pairs qui exécutent les Web services suivants dans une composition-P2P.

Pairs en réserves : les pairs qui peuvent remplacer les pairs suivants dans une composition-P2P.

État de la compo: l'état d'une composition-P2P. Cet attribut décrit la situation d'une composition-P2P à un moment donné : la composition est active si tous les pairs participants sont tous connectés et tous les services de base sont disponibles. Durant l'absence d'au moins un pair participant, toutes les compositions-P2P où il a déjà participé seront non actives. Pour cela, il faut que les autres pairs participants soient informés pour modifier localement l'état du service.

Les attributs *Pairs précédents*, *Pairs suivants*, *Pairs en réserves* et *état de la composition*, sont utilisés pour réparer le chemin de composition en cas où un pair participant quitte le réseau. Ce point sera présenté dans la section 4.2.5.

Pour mieux expliquer le concept de la table de composition et son rôle dans l'algorithme épidémique présenté précédemment, nous présentons un exemple d'un Web service composite qui calcule le prix d'un livre en Dinar Algérien (DA). Ce Web service est composé de quatre services basiques distribués sur le réseau (figure 3.2), chacun d'eux est déployé par un pair (P2, P4, P5 et P7 respectivement).

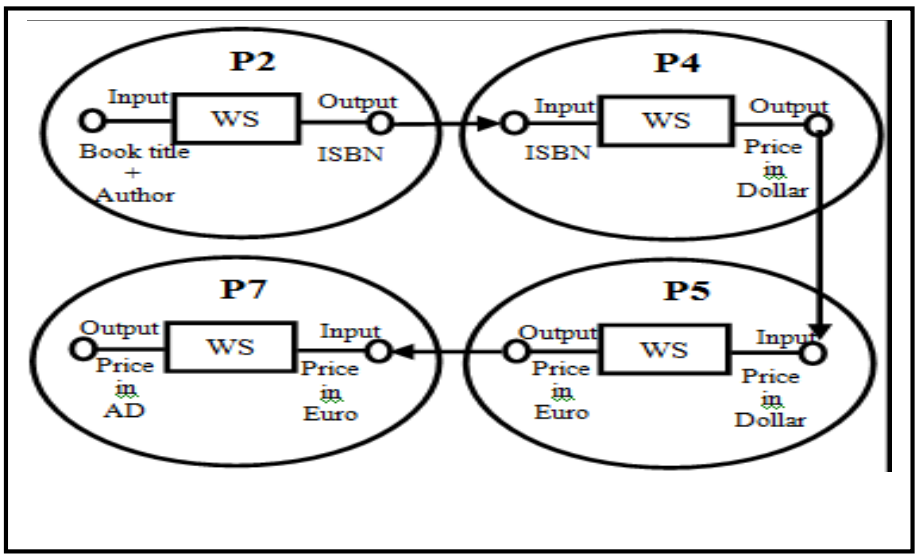


Figure 3.2 : Exemple d'un Web service composite calculant le prix d'un livre en DA.

Suivant cet exemple, la ligne encadrée dans la table 3.2 présente la composition-P2P sauvegardée par le pair P5 (figure 3.2 et 3.3). Ainsi que, la figure 3.3 explique l'opération de la découverte des différents Web services basiques en appliquant l'algorithme épidémique présenté précédemment. Dans ce scénario, le pair P1 reçoit une requête (d'un pair demandeur)

cherchant un Web service qui a en entrée le titre et l'auteur d'un livre et affiche son prix en DA. Le pair P1 transfert cette requête à ces voisins directs parce qu'il ne possède aucune réponse à cette requête. Le pair P2 (pair initiateur) dispose d'un Web service qui a une entrée similaire à celle du Web service demandé (titre et auteur d'un livre), alors que, la sortie du Web service de P2 est l'ISBN du livre. Dans ce cas, le pair P2 remplace l'entrée de la requête et la retransmettre vers ses voisins. Suivant cette méthode, le requête sera propagée jusqu'au pair P7 qui obtient la sortie finale demandée. Cette méthode de découverte est implémentée par l'algorithme épidémique décrit dans la section précédente.

Pair-Initiateur	ID-Compo	Init-input	Init-output	But	Services exécutés	Services réserves	Pairs précédents	Pairs suivants	Pairs réserves	Etat de la compo
P11	0001	….	…..	……	……..	……	…….	……	……..	……
P2	0003	Titre Auteur	Prix	Calculer le prix du livre in DA	WS2	WS3	P4	P7	P6	Active
….	……	…….	……	……	……...	……..	……..	…….	…….	…….
P23	0010	…….	……..	…….	……..	…….	……..	…….	……..	…….

Tableau 3.2 : Contenu de la table de composition (exemple du pair P5 –figure 3.3-)

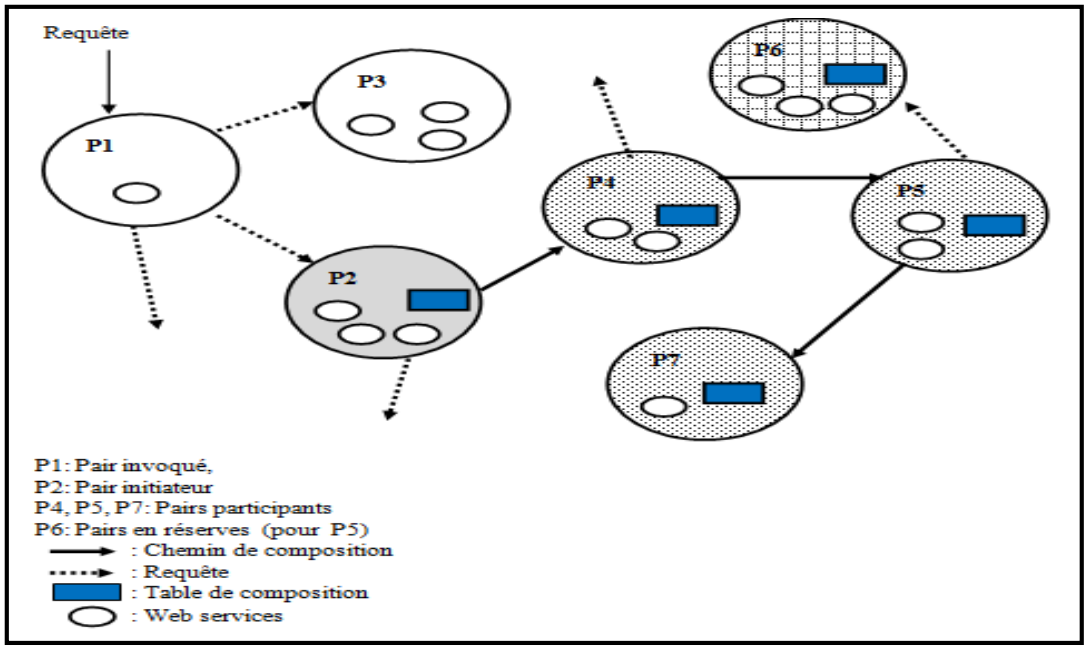


Figure 3.3: Exemple d'une composition dans un réseau P2P non structuré.

3.3 Cohérence des données de la table de composition

La table de composition présente une solution répartie qui permet de réutiliser les Web services déjà composés dans un réseau de système Pair-à-Pair non structuré. Cette répartition peut susciter un problème de cohérence des données de la table de composition dû à la volatilité des nœuds dans un réseau Pair-à-Pair **[GHA+08]**.

Dans la deuxième phase de cette étape de notre travail, nous avons proposé un mécanisme qui assure la cohérence des données tout en réparant le processus des Web services composés. De ce fait, notre approche exploite les caractéristiques des réseaux Pair-à-Pair non seulement pour la découverte des Web services, mais également pour une composition dynamique et réutilisable **[GHA+08]**.

3.3.1 Algorithme de notification

Pour assurer la cohérence des données réparties dans la table de composition, chaque pair participant doit informer ses voisins (tous les pairs précédents et les pairs suivants) de toutes les compositions modifiées. Les autres pairs participants seront informés par inondation dans les deux sens (vers les pairs initiateurs et les derniers pairs des compositions). Ce message est envoyé seulement aux pairs qui ont des compositions communes avec le pair qui a quitté le réseau (figure 3.4).

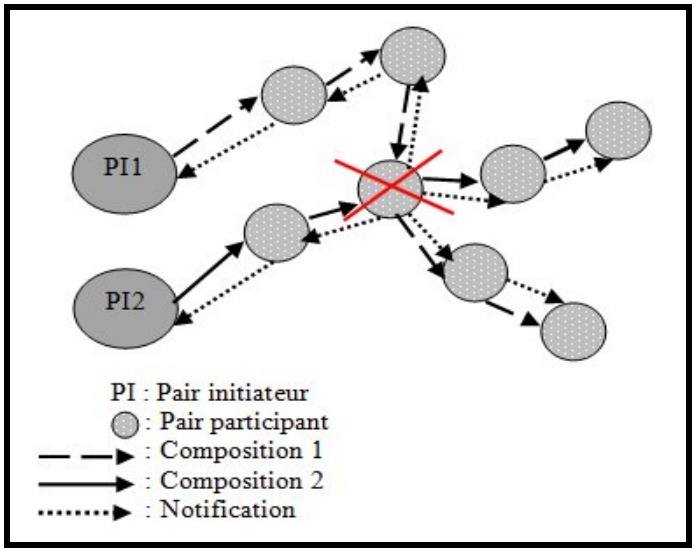


Figure 3.4 : Volatilité des pairs interrompant le processus de composition.

Pour cela, chaque pair exécute l'algorithme suivant lorsqu'il reçoit un message d'un pair qui apparait comme étant pair suivant (Pair_Suivant_I : line 1 de l'algorithme Notification_précédent) :

Notification_précédent

1. Liste_des_précédents= Sélectionner « Pair-Initiateur, ID-compo, Pairs précédents » de la table de composition où le pair suivant = Pair_suivant_I
2. Envoyer à tous les pairs précédents de la Liste_des_précédents «Pair-Initiateur, ID-compo, état de la compo= non active»

Si un pair reçoit un message de notification d'un pair qui apparait comme étant pair précédent (Pair_précédent_I : line 1 de l'algorithme *Notification_suivant*), il exécute l'algorithme suivant :

Notification_suivant

1. Liste_des_suivants= Sélectionner « Pair-Initiateur, ID-compo, Pairs suivants » de la table de composition où le pair précédent= Pair_précédent_I
2. Envoyer à tous les pairs suivants de la Liste_des_suivants «Pair-Initiateur, ID-compo, état de la compo= non active»

3.3.2 Réparation d'une composition

Comme nous l'avons déjà mentionné, les processus correspondants à des Web services composites sont interrompus lorsqu'un pair participant quitte le réseau. Ces processus peuvent être réparés par les pairs précédents du pair perdu. Dans ce cas, un pair précédent essaie de réparer les compositions concernées. Pour une composition donnée, le pair précédent contacte un des pairs en réserve (tête de liste) et il lui envoie une requête qui contient les informations suivantes : Pair-Initiateur, Init-Input, Init-Output, But et l'adresse du pair suivant. Si le pair en réserve obtient un Web service qui a une entrée, une sortie et un but similaire à ceux qu'il a reçu dans la requête, il contacte le pair suivant pour évaluer la composition partielle. Dans le cas contraire, le pair précédent envoie la requête à un autre pair en réserve. Après la réparation de la composition, le pair précédent et le pair suivant mettent à jour leurs tables de composition et exécutent le processus de notification pour informer les autres pairs participants. Ces derniers modifient l'état du Web service correspondant.

Dans le cas où un pair initiateur quitte le réseau, il est nécessaire que son pair suivant déclenche un processus de découverte d'un Web service qui dispose d'une entrée similaire à l'entrée initiale et une sortie identique à l'entrée du service exécuté par ce pair (le pair suivant).

Il est important de signaler que la procédure de réparation n'est pas toujours possible dans une telle architecture. De plus, le processus de notification défini précédemment est plus couteux lorsque le nombre de compositions sauvegardées dans la table est important.

4. Discussions

Comme toute solution, notre proposition présente des avantages et inconvénients que nous pouvons les résumer comme suit :

4.1 Avantages de l'approche

Au lieu d'utiliser un référentiel centralisé des buts réalisés dans le réseau (comme cela a été mentionné dans le travail de **[MAN+07]**), la table de composition présente une solution totalement distribuée qui possède plusieurs avantages :

- Chaque pair participant sauvegarde son historique des compositions et peut exploiter les expériences des autres pairs.
- Le pair qui participe dans plusieurs compositions dispose d'une table très riche, ce qui lui donne plus de possibilités de découvrir des Web services composites pour des futures requêtes. Cette propriété pousse les pairs du réseau de collaborer avec les autres participants, pour accomplir des buts communs. De plus, cette propriété permette d'isoler les pairs égoïstes dans le réseau.
- La table de composition permet de créer des espaces collaboratifs. A fur et à mesure, les pairs qui ont des tables similaires peuvent êtres regroupés dans des communautés.

4.2 Inconvénients

Cependant la solution proposée pose deux problèmes principaux :

- *Problème de la Qualité de Services* : la nécessité de la définition d'un ensemble de métriques permettant la sélection des services découverts.
- *Problème de non considération du parallélisme :* nous avons préféré le parcours du réseau en profondeur pour interroger le maximum des pairs du réseau au lieu de proposer un mécanisme qui permet la découverte et l'exécution des services en parallèle.

4.3 Orientation possible vers le modèle superPair

Dans une architecture Super-pair, les pairs peuvent s'organiser en groupes classés selon la nature et le domaine d'application des Web services fournis. Dans ce cas, les tables de

composition peuvent êtres gérées par les supers pairs. Lorsqu'un pair veut quitter le réseau, il informe seulement son super pair. Ce dernier tente d'abord de réparer la composition par l'utilisation d'un autre Web service qui appartient à un des pairs du groupe (les pairs du groupe ont des services du même domaine). En cas d'échec, le superPair informe les autres superPairs pour modifier les états des Web services composites endommagés à cause de l'absence du pair qui a quitté le réseau (figure 3.5).

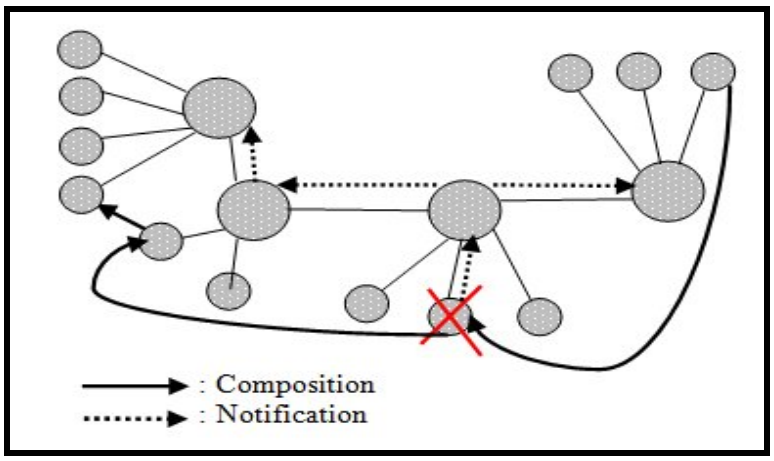


Figure 3.5 : Mécanisme de notification dans un modèle SuperPair.

Il est à noter que la complexité du processus de notification dans un modèle superPair est moins coûteuse que celle de la solution précédente. Le nombre de messages de notification dans ce modèle est N (pour chaque composition) où N est le nombre de superPairs. Ce nombre augmente si les pairs ont l'alternative de s'inscrire dans plusieurs superPairs.

Dans ce modèle, nous pouvons encore améliorer le processus de notification de notre approche si les superPairs sauvegardent dans leurs tables tous les pairs participants pour une composition donnée. Dans ce cas, si un pair participant (P_j par exemple) quitte le réseau, son superPair informe ses voisins que le pair P_j est absent pour qu'il soit remplacé par un autre pair. Les autres superPairs modifient alors les données de toutes les compositions où P_j apparait comme étant un pair participant.

5. Complexité et convergence de l'algorithme épidémique proposé

Nous avons déjà mentionné dans le premier chapitre que les différentes architectures de réseaux P2P montrent des avantages et des inconvénients. Notre stratégie, présentée dans ce chapitre, implémente initialement un algorithme épidémique de découverte des Web services dans un réseau P2P non structuré. De ce fait, la solution proposée possède des caractéristiques qui découlent de la typologie du réseau utilisée d'une part, et des algorithmes de recherche par

inondation d'autre part. Ainsi, en termes de complexité algorithmique, le mécanisme décrit par notre stratégie est très proche de celui implémenté par le protocole Gnutella 0.4, notamment en ce qui concerne les propriétés suivantes :

- Recherche par contenu : la recherche se fait à l'aide d'un mot clé (ou plusieurs) sur les métadonnées de l'objet recherché (dans notre cas la description sémantique du Web service),
- Requêtes dotées par un TTL,
- Chaque requête possède un ID global unique: les nœuds gardent trace des requêtes déjà traitées pour éviter les boucles.

Selon N. Sarshar et al. **[SAR+04]**, les réseaux P2P non structurés (typiquement Gnutella) sont des réseaux « Scalefree » et « Powrlaw », c'est-à-dire possèdent un degré de distribution exponentiel où la probabilité qu'un nœud choisi au hasard ait un degré k est $P(k) \approx k^{-\lambda}$ avec $2 < \lambda < 3$. Ce type de réseaux possède les propriétés suivantes :

- Pour une recherche exhaustive, il faut parcourir tous les nœuds du réseau (inondation). La complexité approche dans le pire des cas $O(n)$ messages où n est le nombre des nœuds.
- Le degré moyen est en $O(\text{Log } n)$, où n est le nombre de nœuds.

Toutefois, notre solution dispose d'un ensemble de caractéristiques qui lui permet d'améliorer la complexité au pire et la complexité en moyenne. Le facteur de réplication de la table de composition joue un rôle important dans ce sens : la probabilité de trouver une composition, dans une table de composition quelconque lors de la recherche d'un Web service, peut diminuer le degré de la complexité en moyenne et au pire de notre algorithme. Valider cette hypothèse nécessite un travail supplémentaire (en cours de réalisation) sur certaines étapes à savoir :

- La simulation qui permet d'évaluer les performances de notre stratégie,
- L'enrichissement des algorithmes proposés avec une approche probabiliste afin de filtrer les pairs pertinents du réseau,
- La vérification de la gestion de la terminaison de l'algorithme proposé.

6. Conclusion

Les avantages apportés par les technologies du Web sémantique à l'opération de découverte des Web services sont bien réels mais un frein leur est imposé par l'UDDI qui ne supporte que la recherche basée sur des informations de haut niveau spécifiques aux

entreprises et aux services. En effet, UDDI ne reçoit pas les spécificités des capacités des services pendant le mapping sémantique (Machmaking) et ne saisit pas les relations entre les entités dans son répertoire et n'est donc pas capable de faire usage de l'information sémantique pour déduire les relations en cours de la recherche.

Pour régler ces problèmes, les méthodes de découverte distribuées sont devenues une nécessité. Dans ce chapitre, nous avons présenté une solution totalement distribuée pour la découverte et la composition des Web services. Nous avons employé les réseaux P2P non structurés pour implémenter un algorithme épidémique qui permet la recherche, le matchmaking et la composition des services pour répondre à une requête spécifique.

Afin d'accélérer le temps de recherche et d'optimiser le passage à l'échelle dans le réseau, nous avons implémenté une table distribuée permettant de préserver la trace des compositions déjà réalisées dans le réseau. Cette table nous permet de réutiliser les expériences des différents pairs participants, et de créer un espace de collaboration entre les pairs fournissant des Web services.

Dans le chapitre suivant, nous allons présenter une architecture permettant l'implémentation de la stratégie présentée dans ce chapitre.

Chapitre 4

Une architecture distribuée pour la découverte et la composition des Web services

1. Introduction

La convergence entre les Web services, le Web sémantique et les systèmes P2P a fait l'objet de plusieurs travaux de recherches. Dans ce contexte, la découverte des Web services sémantique dans les systèmes présente une partie intéressante pour laquelle sont proposés différentes architectures et frameworks.

Dans notre travail, et après avoir présenté la stratégie de la découverte des Web services sémantique dans un réseau P2P non structuré, nous allons présenter dans ce chapitre une architecture permettant l'implémentation de cette stratégie.

Nous allons décrire tout d'abord une architecture globale qui comporte un point central pour créer une ontologie de référence ainsi qu'un framework distribué qui permet d'implémenter l'algorithme épidémique déjà présenté dans le chapitre 3. Nous détaillons les différents composants de ce framework, ainsi que toutes les interactions possibles.

Ces descriptions sont suivies de la présentation d'un processus de développement permettant l'amélioration du fonctionnement de l'architecture proposée avant de terminer ce chapitre par l'exposition de quelques travaux voisins.

2. Architecture de référence

Dans cette section, nous décrivons l'architecture globale pour implémenter la stratégie proposée dans le chapitre précédent. L'élément clé de cette architecture est le framework distribué PM4SWS (P2P-Based Model for Semantic Web Services) **[GHA+11]**. Ce dernier est installé sur les différents pairs du réseau, alors qu'une base centrale des ontologies OWL (et éventuellement des descriptions OWL-S) est utilisée comme une référence pour le développement des ontologies locales. L'architecture globale est présentée dans la figure 4.1.

Dans cette architecture, chaque pair dans le réseau implémente un ensemble de Web services décrits sémantiquement en OWL-S. Ce dernier est conçu comme le langage sémantique le plus utilisé dans la communauté du Web sémantique. Cependant, pour éviter l'incompatibilité qui peut être créée à cause de l'utilisation des concepts hétérogènes d'un pair à un autre, nous proposons l'utilisation d'une base ontologique commune qui contient une ontologie globale des différents domaines de développement des Web services. Cette base fournit un ensemble d'ontologies directement téléchargeable par les différents pairs. Elle peut aussi être utilisée pour développer des ontologies locales au niveau de chaque pair. De plus,

cette base est évolutive : elle peut être enrichie par des nouveaux concepts (ou ontologies) proposés par les différents pairs.

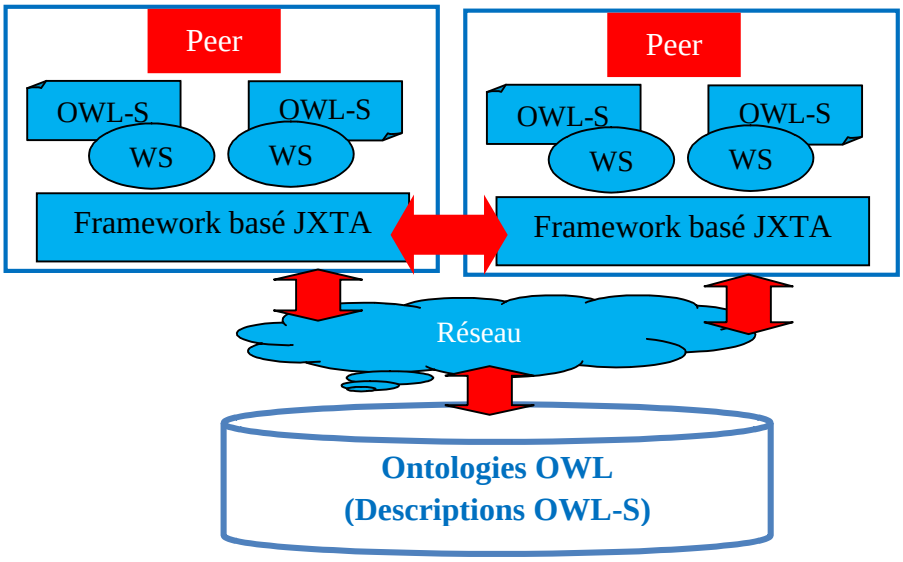


Figure 4.1: Architecture de référence.

De ce fait, les ontologies proposées par la base centrale seront utilisées pour développer les descriptions sémantiques (localement) des différents Web services proposés par les pairs du réseau. Cette architecture nous permet d'utiliser le même langage de description sémantique (OWL-S) en utilisant le même langage d'ontologie (OWL) et en se servant au maximum possible les mêmes concepts des domaines.

Dans ce contexte, un exemple d'une ontologie universelle a été développé et publiée par Ganjisaffar and Saboohi **[GAN+06]**. En plus de cette ontologie, plus de 240 descriptions sémantiques OWL-S ont été proposées par le créateur de cette initiative. Ces différentes descriptions ainsi que l'ontologie globale sont libres au téléchargement et à l'utilisation pour les développeurs des Web services sémantiques.

Dans ce qui suit, nous décrivons l'architecture détaillée du framework PM4SWS ainsi que ses différents composants de base.

3. Architecture détaillée

L'idée principale du framework PM4SWS est de fournir une découverte et une composition des Web services purement décentralisées. Dans ce contexte, la table de composition de chaque paire participant présente une mémoire cache qui sauvegarde les traces des différentes compositions déjà réalisées avec succès. Ce framework contient une

interface utilisateur et trois couches principales : Gestionnaire de la sémantique, Module de la composition locale et Module de la composition P2P (figure 4.2) **[GHA+10]**.

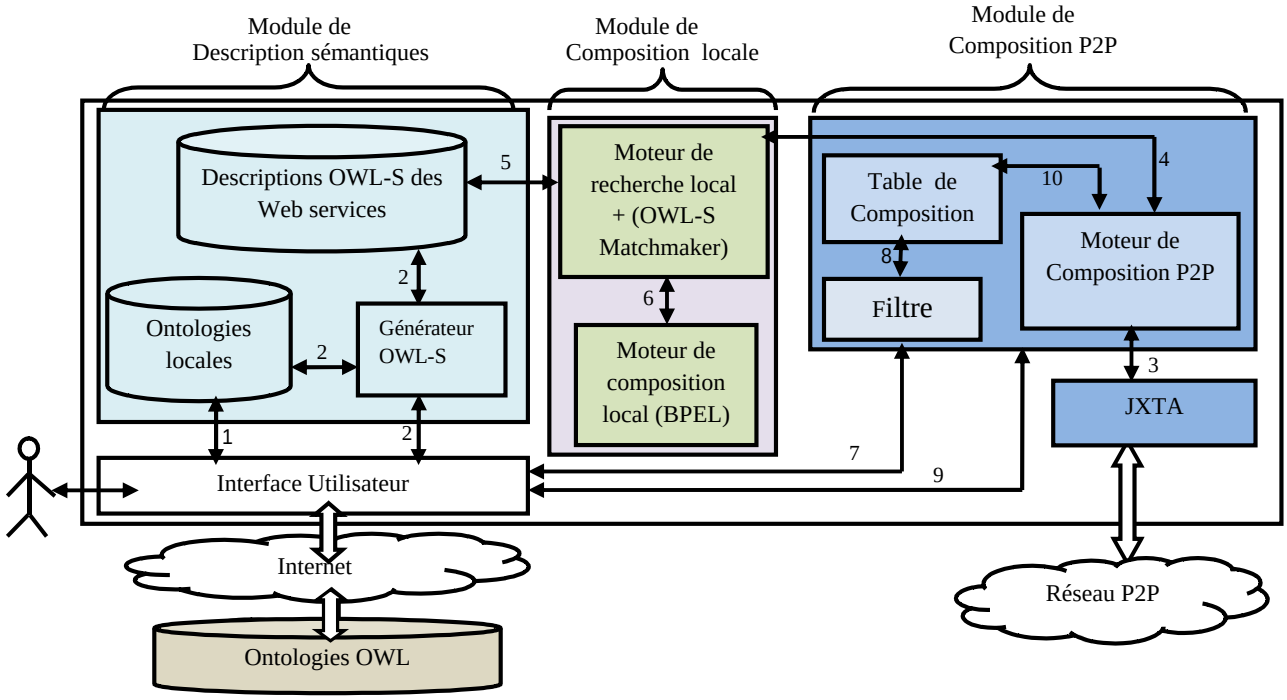


Figure 4.2 : Architecture détaillée de PM4SWS.

3.1 Interface utilisateur

L'interface représente l'intermédiaire ente l'utilisateur le framework. Elle peut être utilisée pour plusieurs tâches :

- Télécharger des ontologies et des descriptions OWL-S au niveau de la base centrale
- Proposer des modifications aux nouvelles ontologies à la base centrale
- Générer des descriptions OWL-S en utilisant le générateur OWL-S (flèche 2, figure 4.2)
- Nettoyer et filtrer la table de composition en utilisant le composant filtre (flèches 7 et 8, figure 4.2).
- Envoyer des requêtes aux autres pairs du réseau (flèche 9, figure 4.2).

Il est intéressant de signaler que l'interface utilisateur peut être implémentée sous forme d'un portail qui offre plusieurs portlets, chacune est spécifiée à domaine bien défini.

3.2 Module de description sémantique

Cette couche gère les descriptions sémantiques des Web services. L'utilisateur utilise un générateur OWL-S et une ontologie OWL locale pour générer les descriptions OWL-S des

différents Web services. L'ontologie locale peut être développée localement selon l'ontologie centrale. Dans d'autres cas, l'utilisateur peut télécharger directement une (ou plusieurs) ontologie(s) de domaine à partir de la base centrale.

3.2.1 Descriptions OWL-S des Web services

L'utilisateur peut télécharger des descriptions OWL-S prédéveloppés au niveau de la base centrale (**[GAN+06]** fournissant ainsi plus de 240 descriptions OWL-S dédiées aux Web services des différents domaines). Il est important de mentionner que dans ce cas, le développeur peut utiliser une description OWL-S comme un Template pour développer son Web service.

Dans la deuxième possibilité, le développeur peut générer des descriptions OWL-S en utilisant le générateur OWL-S et la base d'ontologie locale.

3.2.2 Ontologies locales

Comme nous l'avons déjà mentionné, l'utilisateur peut télécharger les ontologies qui conviennent aux services fournis par son nœud. De plus, le développeur peut localement mettre en place des ontologies OWL en utilisant un environnement dédié comme Protégé OWL par exemple **[Protégé]**. Ce dernier est une plateforme open source pour développer des ontologies et des systèmes à bases de connaissances.

3.2.3 Générateur OWL-S

Le générateur OWL-S utilise la description WSDL du service et une ontologie de domaine pour générer la description OWL-S. Dans notre travail, nous avons utilisé l'outil wsdl2owl-s **[wsdl2owls]** pour implémenter ce composant.

3.3 Module de composition locale

L'objectif de cette couche est de découvrir ou de composer un Web service local qui répond à des requêtes externes (des autres pairs). Pour atteindre ce but, nous proposons deux composants principaux dans cette couche : le moteur de recherche local et moteur de composition local.

3.3.1 Moteur de recherche et l'OWL-S Matchmaker

Les deux principales tâches de ce moteur concernent la recherche des Web services basiques ou une éventuelle composition locale (Web service composite).

Lorsque il reçoit une requête à partir du réseau (flèche 3, figure 4.2), le moteur de composition P2P passe la requête au moteur de la recherche local (flèche 4, figure 4.2). Ce dernier cherche un Web service basique qui répond à cette requête (étape 2 de l'algorithme principal). En même temps, le moteur utilise un OWL-S Matchmaker pour découvrir une composition possible à partir d'un ensemble de Web services basiques.

Il y a trois scénarios possibles:

- Si le moteur ne trouve aucun Web service (basique ou composé) qui répond à la requête, il retourne une réponse négative au moteur de composition P2P.
- S'il y a un éventuel basique ou composite Web service, il retourne une réponse positive. Dans ce cas, le moteur de recherche génère un fichier BPEL qui sera utilisé par le moteur de composition local (flèche 6, figure 4.2). Dans l'étape de l'invocation, Ce dernier invoque les Web services selon le processus défini dans le fichier BPEL.
- S'il y a une semi-composition (formée d'un ou plusieurs Web services basiques), le moteur de recherche génère une requête pour la proposée au moteur de composition P2P qui peut alors continuer l'étape de découverte en envoyant la requête proposée aux pairs du réseau.

Il est important que le phénomène de l'algorithme épidémique soit expliqué par ce dernier scénario : tous les pairs exécutent les mêmes étapes lorsqu'ils reçoivent une requête à partir d'un autre pair. C'est l'objectif principal du framework distribué qui permet de fournir une plateforme collaborative entre les pairs du réseau.

3.3.2 Moteur de composition local

Le moteur de composition du framework PM4SWS est un moteur BPEL. Nous avons choisi ce langage parce qu'il est mature. De plus, les moteurs BPEL existants génèrent des processus abstraits et exécutables. Le fichier BPEL généré automatiquement, peut être utilisé par le moteur d'invocation (BPEL invoker) du moteur de composition local, pour exécuter un Web service composé localement ou dans le réseau.

3.4 Module de composition P2P

Ce module présente une interface entre le pair et le réseau P2P. C'est le composant principal qui gère la communication et permet d'établir des compositions collaboratives entre les différents pairs participants. Cette couche contient trois (3) composants : la table de composition déjà présentée dans le chapitre précédent, le filtre et le moteur de composition P2P.

3.4.1 Moteur de composition P2P

Ce moteur contient un composant requête/réponse pour envoyer et recevoir les requêtes (flèche 9, figure 4.2). Ce composant est utilisé dans plusieurs scénarios possibles qui ont une relation avec ceux déjà présentés dans la section précédente (moteur de recherche et OWL-S Matchmaker). De plus, l'utilisateur fait appel au moteur de composition P2P pour lancer une découverte dans le réseau.

Lorsque il reçoit une requête à partir d'un autre pair du réseau (flèche 3, figure 4.2), le moteur de composition P2P transfert la requête au moteur de composition local qui peut retourner une des réponses suivantes:

- *Réponse positive* : dans ce cas le moteur de composition P2P retourne la réponse au pair demandeur.
- *Réponse négative* : le moteur de composition P2P évalue un nouveau TTL et transfert la requête à ses pairs voisins dans le réseau.
- *Trouver une composition active dans la table de composition* : dans ce cas le pair retourne l'adresse du pair initiateur de cette composition
- *Proposition d'une semi-composition* : le moteur P2P évalue un nouveau TTL et continue la découverte en utilisant la requête proposée par le moteur de composition local.

3.4.2 Filtre de la table de composition

Le filtre est un composant au moyen duquel l'utilisateur peut nettoyer la table de composition de toutes les compositions inactives (flèche 7, figure 4.2). Cette opération est très importante dans un environnement dynamique comme les réseaux P2P où les pairs peuvent quitter et rejoindre le réseau fréquemment. Pour cette raison, plusieurs compositions deviennent inaccessibles lorsqu'un (ou plusieurs) pair(s) participant(s) quitte(ent) le réseau. Le filtre réalise des statistiques concernant les entrées de la table de composition pour détecter les compositions inactives depuis un certain temps (flèche 8, figure 4.2). Ces statistiques offrent une vision claire sur les états des compositions dans la table.

Il est important de noter que la nature volatile des nœuds créé un problème de cohérence des données de la table. Ce problème a été réglé par les algorithmes de notification présentés dans le chapitre précédent.

4. Processus pour améliorer le fonctionnement de l'architecture

Suivant le fonctionnement du framework PM4SWS, nous pouvons déduire que le temps de découverte est jugé important à cause du matching sémantique (OWL-S matchmaking) réalisé en temps d'exécution. Pour cette raison, nous allons présenter dans la section suivante un processus de développement pour améliorer la solution proposée. Ce processus est divisé en trois phases : en temps passif, en temps d'exécution et la réutilisation des expériences.

4.1 Au temps passif (Build time)

Les étapes suivantes sont réalisées en temps passif pour réduire le temps de découverte, spécialement le matchmaking sémantique. Ces étapes sont réalisées par chaque fournisseur de Web services (pair).

4.1.1 Développement des ontologies locales

Comme nous l'avons déjà vu, le développement des ontologies locales est réalisé suivant celui de la base centrale. Le fournisseur utilise les ontologies correspondantes aux domaines des Web services offerts par ce fournisseur. Dans d'autres cas, le développeur peut télécharger et réutiliser des ontologies de la base centrale sans aucune modification.

Pour développer des ontologies OWL, plusieurs environnements sont disponibles. Dans notre cas, nous avons utilisé Protégé OWL comme nous allons le montrer dans le chapitre suivant.

4.1.2 Génération des descriptions OWL-S

Dans le premier cas, le développeur peut réutiliser directement quelques fichiers OWL-S fournis par la base centrale. Par exemple, **[GAN+06]** fournit environ 240 descriptions OWL-S pour une large variété de Web services. Dans ce cas, le développeur doit implémenter ses Web services suivant les descriptions OWL-S.

Dans une autre possibilité, le développeur peut générer une description OWL-S à partir de la description WSDL du Web service et en utilisant l'ontologie de domaine correspondante. Pour notre part, nous recommandons l'utilisation de certains outils libres comme le convertisseur wsdl2owl-s **[wsdl2owls]**, OWL-S editor **[owls-editor]** et l'API OWL-S pour OWL. La figure 4.3 et 4.4 présentent un exemple d'utilisation de l'outil wsdl2owls pour la génération d'une description OWL-S d'un Web service conçu pour la réservation d'un vol entre de villes.

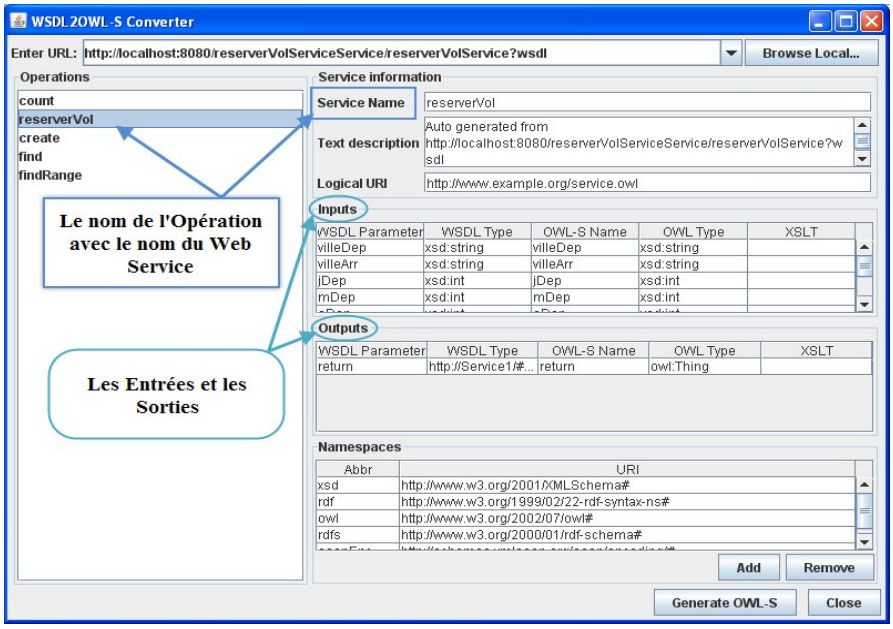


Figure 4.3: le convertisseur wsdl2owl-s

Si la génération de la description OWL-S a été terminée avec succès, le message suivant sera affiché (figure 4.4).

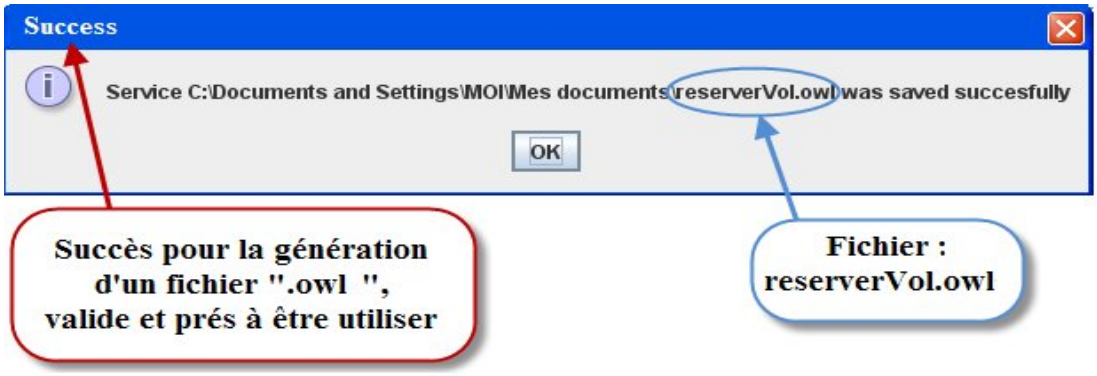


Figure 4.4 : Génération d'une description OWL-S avec succès.

4.1.3 Recherche des éventuelles compositions

Chaque pair participant implémente un ensemble de Web services basiques qui peuvent répondre aux requêtes reçues du réseau. Cependant, dans plusieurs cas, un seul Web service ne peut pas garantir la réponse complète. De ce fait, la composition de plusieurs Web services basiques locales (appartenants au même pair) peut fournir la réponse attendue.

Dans la solution déjà présentée, la recherche d'une composition locale qui répond à la requête se fait en temps d'exécution bien que cette opération puisse consommer un temps important à cause de l'opération du matchmaking sémantique effectuée entre les descriptions OWL-S des Web services candidats (pour une composition). De plus, cette opération peut

échouer et augmenter la probabilité de la consommation du TTL associé à la requête, ce qui engendre une réponse défavorable avant même de consulter les autres pairs du réseau.

Pour cette raison, il s'avère préférable de rechercher d'éventuelles compositions en temps passif. Pour réaliser cet objectif, nous proposons l'utilisation d'un OWL-S Matchmaker qui utilise les descriptions OWL-S des Web services pour vérifier s'ils sont composables ou non. Dans notre travail, nous avons utilisé OWLS-MX 2.0 **[owls-mx]** qui est un outil libre permettant de composer (abstraitement) sémantiquement des Web services en utilisant ces descriptions OWL-S. La figure 4.5 présente l'interface principale de l'outil OWLS-MX 2.0.

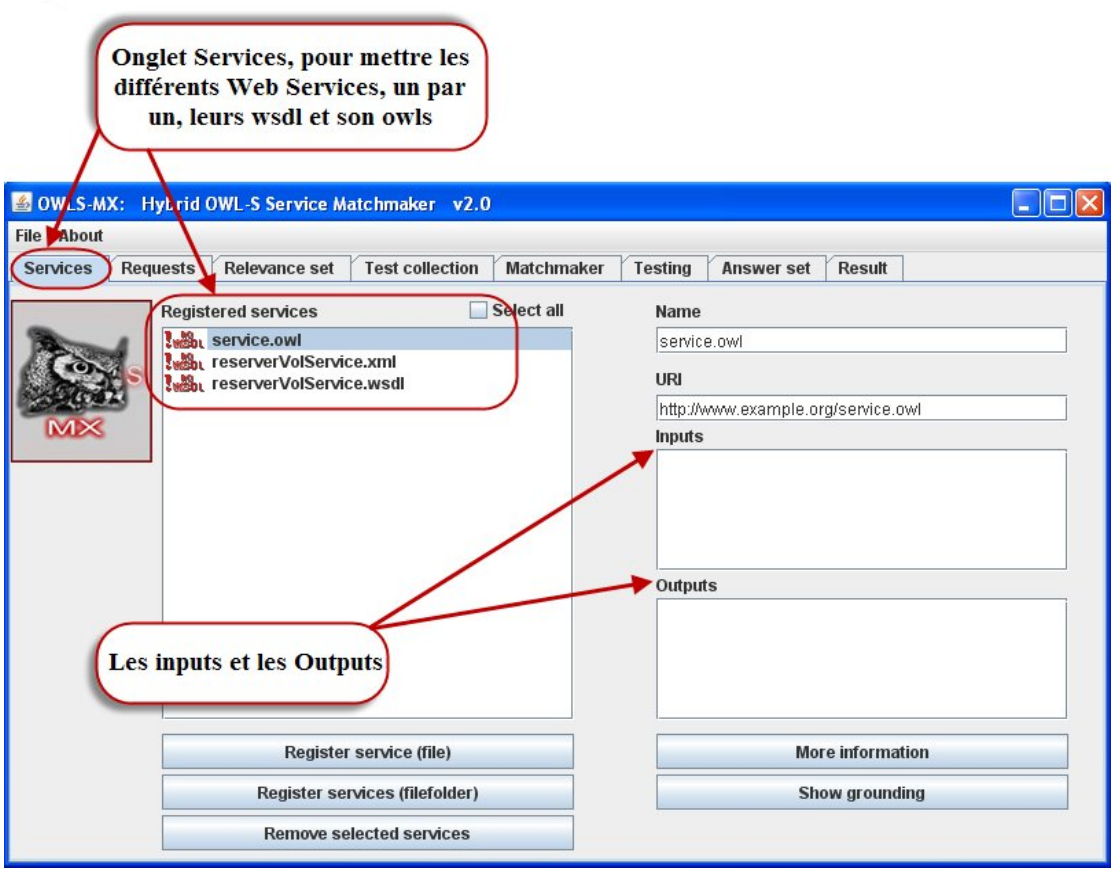


Figure 4.5: OWL-S MX Matchmaker.

OWLS-MX 2.0 accepte en entrée un ensemble de descriptions OWL-S correspondantes un ensemble de Web services, après une opération de matchmaking, il donne comme résultat un boolean qui indique la possibilité de composer ces Web services ou non.

4.2 Au temps d'exécution (Runtime)

Cette phase contient deux étapes principales : la découverte des Web services distribués sur le réseau et l'invocation des Web services élus pour la composition.

4.2.1 Etape de découverte des Web services

Cette étape est implémentée par l'algorithme épidémique présenté dans le chapitre précédent en choisissant l'un des types de chainages proposés (en avant ou en arrière).

4.2.2 Etape de composition (en utilisant un moteur BPEL)

Si l'étape de découverte a été terminée avec succès (un message est envoyé au pair demandeur), le pair initiateur lance l'étape de composition par la génération d'un fichier BPEL à partir du résultat retourné qui comporte l'ensemble des pairs participants avec l'ordonnancement et les URL des contrats WSDL des Web services candidats (proposés par les pairs participants). Ensuite, le pair demandeur peut envoyer les données d'invocation qui seront utilisées par le pair initiateur pour lancer l'exécution du fichier BPEL correspondant en utilisant le moteur BPEL local.

Il est très important de signaler que même si la découverte a été réalisée d'une manière sémantique (en utilisant les descriptions OWL-S des Web services), nous sommes obligés de composer les Web services découverts en utilisant un autre standard de composition comme BPEL. Ceci résulte du fait que OWL-S peut décrire un processus métier à l'aide de « OWL-S Process », mais malheureusement cette description reste abstraite parce que aucun environnement pour le moment ne fournit un moteur d'exécution des processus OWL-S.

4.3 Réutilisation des expériences

La réutilisation des expériences est une étape primordiale dans notre solution, parce qu'elle présente un facteur important pour diminuer le temps de découverte des Web services. Dans notre cas, nous avons présenté la réutilisation des expériences par la création de la table de composition implémentée au niveau de chaque pair participant. Ce type d'expérience est connu comme étant local et propre à chaque pair. Cependant, l'algorithme épidémique de découverte offre un mécanisme de collaboration implicite permettant à chaque pair de réutiliser ces expériences ainsi que les expériences des autres pairs.

Dans notre travail, cette tâche peut être réalisée plutôt en temps passif qu'en temps d'exécution. Ce dernier est déjà présenté dans l'étape de découverte à l'aide de la recherche dans la table de composition de chaque pair lorsqu'il reçoit une requête. Dans le temps passif, la réutilisation des expériences s'inscrit dans l'étape de filtrage de la table de composition.

Comme nous l'avons déjà signalé dans le chapitre précédent, cette solution peut être enrichie par le regroupement des pairs qui ont des tables de composition similaires dans des clusters. Dans ce cas, nous pouvons créer un point commun qui regroupe les expériences des

pairs d'un cluster dans une base centrale implémentée par le super-Pair. Cette base sera très utile pour chercher des compositions déjà réalisées dans le cluster avant de lancer une nouvelle opération de découverte.

4.4 Architecture améliorée

Le résultat du processus de développement avec les différentes étapes présentées dans la section précédente est l'architecture améliorée montrée dans la figure 4.6 comportant les interactions possibles entre les différents composants du framework PM4SWS.

Au temps passif

(1) Développement (Modification) des ontologies locales/ formuler une requête
(2) Génération des descriptions OWL-S
(3) Recherche d'éventuelles compositions
(4) Modification/suppression des expériences locales
(5) Téléchargement/ consultation de l'ontologie globale.

Au temps d'exécution

(A) Lancement d'une requête dans le réseau
(B) Recherche d'un Web service (ou une composition locale) pour répondre à une requête/ génération d'un fichier BPEL
(C) Si le pair est initiateur : exécution du fichier BPEL/invocation des Web services
(D) Réutilisation des expériences
(E) Envoi/ Réception des messages (découverte, composition)
(F) Communications P2P

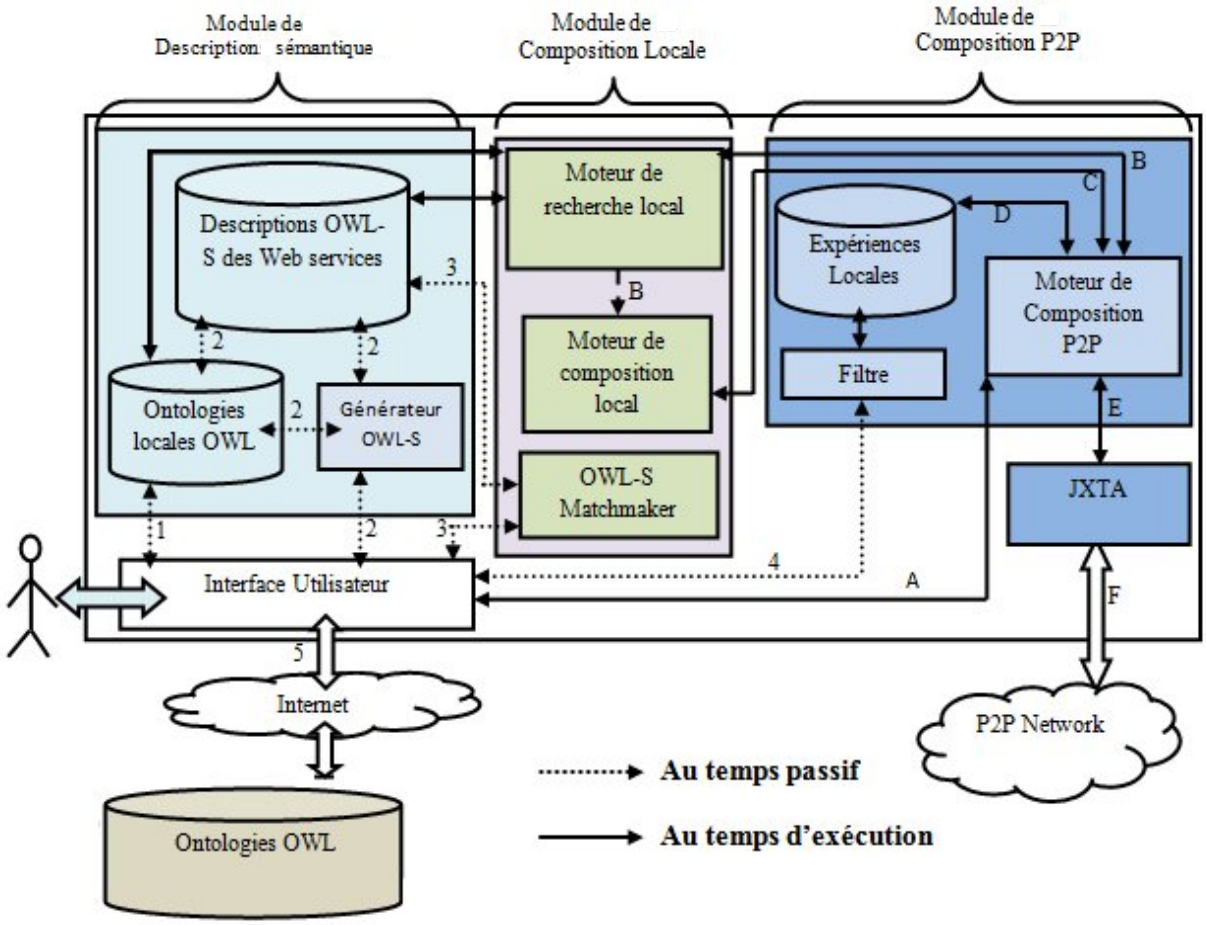


Figure 4.6 : Architecture améliorée de PM4SWS.

5. Travaux voisins

En général, les travaux de recherche qui convergent dans la synergie des domaines de la composition des Web services, des Web services sémantique et des systèmes P2P, peuvent être classés selon les catégories suivantes :

- L'apport des technologies du Web sémantique à la découverte et la composition des Web services **[JIN+05]**, **[BER+05]**.
- L'utilisation des technologies du Web sémantique (spécialement les ontologies) pour chercher et classifier le contenu dans un réseau P2P **[HAA+04]**, **[WAN+07]**.
- La convergence entre les systèmes Multi Agents (SMA) et les réseaux P2P pour la composition des Web services sémantiques **[KUN+06]**.
- L'utilisation des méthodes formelles pour la vérification et la validation des compositions des Web services **[TAN+07]**.
- L'évaluation des performances et de la qualité des Web services composites **[VU+05]**, **[LIU+05]**.

Récemment, plusieurs articles de recherche ont été publiés dans ce contexte. Ils proposent des solutions basées sur les systèmes P2P pour la découverte et la composition des Web services sémantiques. Plusieurs travaux s'intéressent aux méthodes décentralisées (basées sur les réseaux P2P) pour la découverte des Web services. Parmi ces travaux, **[MAN+07]**, **[EME+04]** et **[SAH+05]** ont des concepts similaires à ceux qui sont utilisés dans notre travail.

F. Mandreoli et al **[MAN+07]** présentement le framework FLO^{2}WER qui supporte l'interopérabilité des Web services sémantiques dans les environnements P2P dynamiques et hétérogènes. Ils ont adapté une approche hybride pour exploiter les avantages des registres centralisés, ainsi que la dynamique et l'évolutivité des réseaux P2P structurés.

L'idée principale de FLO^{2}WER est la suivante : au lieu de centraliser les connaissances spécifiques aux services disponibles dans le système, il préserve, d'une manière centralisée, les connaissances correspondantes aux objectifs qui doivent êtres satisfaits dans le réseau. Par conséquent, chaque objectif spécifie un sous réseau de services et est sauvegardé dans un référentiel approprié nommé *Répertoire des buts*.

Néanmoins, les auteurs de ce travail ne décrivent pas comment les objectifs sont découverts. De plus, l'utilisation d'un répertoire central des buts est pratiquement similaire aux méthodes de découverte centralisées : le répertoire central présente un seul point d'échec et au moment de l'exécution, il est difficile de remplacer un pair absent ou de découvrir un Web service équivalent à ce qui est implémenté par le nœud manquant.

En revanche, notre méthode de découverte est totalement décentralisée sans aucun contrôle centralisé. Les Web services déjà composés dans le réseau sont publiés d'une manière distribuée dans toutes les tables de compositions des pairs participants.

De leur côté, **[HU+05]**, **[EME+04]** et **[SAH+05]** définissent un Web service composite comme un automate fini. J. Hu et al **[HU+05]** et F. Emekci et al **[EME+04]** proposent un framework basé sur un réseau P2P structuré pour la découverte des Web services. Ces derniers sont localisés par les fonctionnalités et le comportement du processus. Les auteurs représentent le comportement du processus des Web services par des automates finis et utilisent ces derniers pour publier et invoquer les services.

Dans notre travail le comportement du processus des Web services et représenté par une séquence finie des pairs participants. De plus, dans notre contexte nous avons proposé une solution basée sur les réseaux P2P non structurés où les pairs participants et les Web services candidats ne sont pas connus à l'avance. Ainsi, le mécanisme de publication de notre solution

est implicitement implémenté par la table de composition. Si un pair veut réutiliser un Web service déjà composé, il envoie la requête au pair initiateur de cette composition.

Dans un autre travail, T. Essafi et al **[ESS+05]** présentent une approche P2P pour la découverte des Web services en utilisant les ontologies. Ce travail utilise le mapping entrée/sortie proposé dans l'algorithme de **[PAO+02]**, et étend la solution décrite dans **[PAO+03]** par un mécanisme qui localise les serveurs dans un réseau P2P pour simplifier le routage des requêtes.

Pareillement à la solution présentée en **[MAN+07]**, ce travail adopte les réseaux P2P hybrides alors que notre solution utilise les réseaux non structurés pour assurer une découverte dynamique sans contrôle centralisé. De plus, T. Essafi et al utilisent DAML-S qui est moins flexible que la nouvelle spécification OWL-S.

Dans leur travail exposé dans **[ZHE+09]**, Z. Zhengdong et al. désignent un mécanisme de localisation des Web services sémantiques dans un réseau P2P structuré basé sur le protocole CAN. Dans ce mécanisme, chaque Web service est enregistré dans un nœud spécifique dans le Web. La distribution des Web services sur les nœuds est devisée par des régions partagées par tous les nœuds. Cependant, ce travail considère uniquement les réseaux P2P basés sur le protocole CAN, alors que, notre approche est générique et adaptable à n'importe protocole non structuré.

6. Conclusion

Dans ce chapitre, nous avons décrit une architecture pour implémenter la stratégie présentée en chapitre 3. Cette architecture comprend un framework distribué pour la découverte et la composition des Web services sémantiques. Nous avons détaillé les différents composants de ce framework et les interactions entre eux en faisant référence à l'algorithme de découverte déjà présenté dans le troisième chapitre. Un processus de développement a été ajouté pour améliorer le fonctionnement de cette architecture.

Nous avons aussi proposé l'utilisation d'un référentiel pour la publication d'une ontologie globale au lieu de créer un point central de publication et de découverte des Web services fournis par les pairs du réseau. Cette ontologie est utilisée pour développer les différentes descriptions sémantiques des Web services et nous permet d'assurer l'interopérabilité et l'homogénéité sémantique.

L'avantage de l'ontologie globale est qu'elle est utilisée en temps passif, ce qui n'a aucune influence sur le temps de la recherche de la phase de découverte.

Dans le chapitre suivant, nous allons présenter une étude de cas pour monter l'implémentation des différents composants de l'architecture présentée dans ce chapitre.

Chapitre 5

Etude de cas et implémentation

1. Introduction

Nous venons de détailler dans le chapitre précédent les différents composants de l'architecture qui implémente la stratégie de découverte déjà présentée en chapitre 3. Dans le présent chapitre, nous présentons les résultats obtenus à l'implémentation.

En effet, dans une première partie, nous décrivons une étude de cas. Dans une seconde partie, nous précisons les différents outils techniques utilisés. Nous présentons ensuite l'implémentation des différents composants inclus dans.

2. Présentation de l'étude de cas

Dans cette section, nous voulons présenter l'implémentation de quelques composants du framework décrit dans le chapitre précédent. Cet exemple présente une application distribuée dite « Constantine Books » permettant de chercher des livres. Cette application donne la possibilité à l'utilisateur de chercher le prix d'un livre en US dollar, Euro ou en Dinar algérien. L'utilisateur peut utiliser en entrée plusieurs arguments de recherche comme : l'auteur, le titre du livre, maison d'édition…etc.

Pour implémenter cette application, nous voulons composer plusieurs Web services distribués sur un réseau P2P. Le scénario suivant présente la composition de trois Web services :

- Search-ISBN : Donner l'ISBN d'un livre
- Search-Price : donner le prix d'un livre en US dollar.
- Convertor-Money : convertisseur de monnaie US Dollar=> Euro

Web service	Entrées	Sorties
Search-ISBN	Title **ou** Author **ou** Edition **ou** Year-of-edition	ISBN
Search-Price	ISBN	Price
Convertor-money	Money-USD	Money-EUR

Tableau 5.1 : Entrées/sorties des Web services candidats.

La composition des ces services donne un Web service composite avec les arguments suivants :

Entrée: **bookInfo**=>Title||Author||Edition|| year-of-edition

Sortie: **bookPrice**

But:*input:=#**BookInfo**=>book:Title||Author||Edition||year-of-edition

*output:=#**book:Price**//Money:EUR

Pour simplifier le test, la figure suivante présente l'implémentation des trois Web services collaboratifs sur deux pairs (figure 5.1). Le premier pair (qui est le pair initiateur) déploie deux Web services : "Search-ISBN" and "Search-Price". Le deuxième pair implémente "Convertor-money".

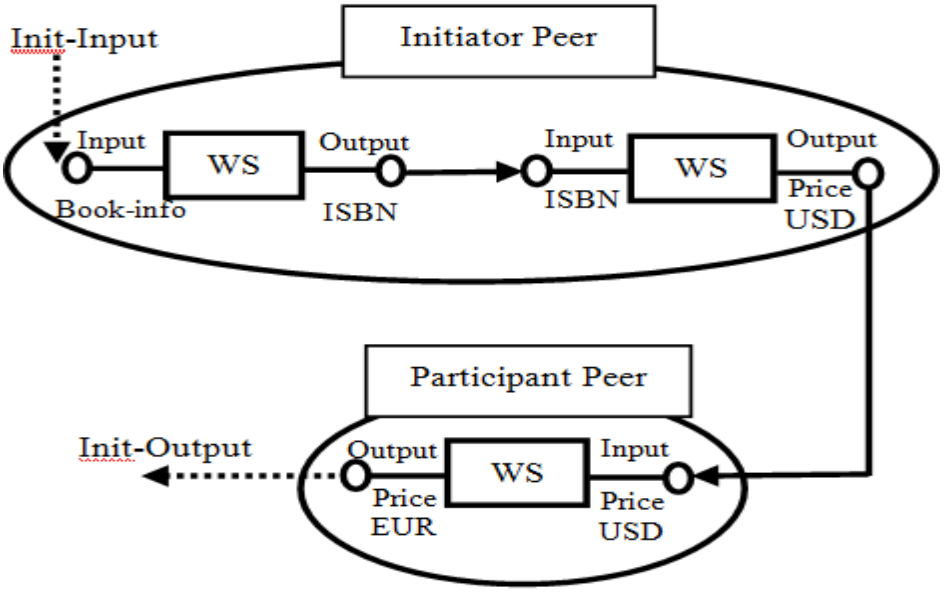


Figure 5.1: Exemple d'un Web service composé par 3 pairs.

Lors du déroulement de l'algorithme épidémique de la découverte, lorsqu'il reçoit la requête, le moteur de composition P2P du pair initiateur passe cette requête au moteur de recherche local. L'annexe A présente l'écriture de la requête suivant le modèle OWL-S. Cette requête faite appel à l'ontologie "Book.owl" (Annexe B).

3. Présentation des outils technologiques utilisés

Dans ce travail nous avons utilisé plusieurs outils techniques qui sont tous « libres » et d'autres sont « open source ». Ces outils sont issus aux technologies du Web sémantique et celles des systèmes P2P. Ils sont présentés comme suit :

a) Langages

- Le langage de développement utilisé est le langage Java dans sa version d'entreprise "Java EE 5 SDK". Le kit Java EE 5 SDK peut être téléchargé à partir du lien : http://java.sun.com/javaee/
- L'autre langage important utilisé pour développer notre application est le langage WS-BPEL 2.0 (ou simplement BPEL), ce dernier a été utilisé pour réaliser la composition des Web services.
- Les autres langages utilisés pour développer notre application sont le langage OWL-DL et OWL-S.

b) Serveur

Actuellement, il y a un grand nombre de serveur disponibles gratuitement sur internet. Cependant, dans notre travail nous avons utilisé le serveur Glass Fish sous sa dernière version V2.1 (connu auparavant par le nom Sun Java System Application Server). Nous avons choisi ce serveur car il est déjà intégré dans Java EE 5 SDK Update 7.

c) Système de gestion de base de données

Nous avons choisi le SGBD Java DB version 10.4 inclus dans la Java SE Development Kit, il est disponible sur le site : http://developers.sun.com/javadb/

d) Moteur d'exécution BPEL

Le moteur d'exécution BPEL que nous avons utilisé est "BPEL Service Engine". Pour les détails d'utilisation et de configuration vous pouvez consulter le site : https://open-esb.dev.java.net/kb/preview3/ep-bpel-se.html

e) Environnement de développement

Pour développer quelques composants de notre architecture, nous avons choisi Netbeans comme environnement de développement dans sa version 6.5 téléchargeable à partir de : http://www.netbeans.org.

f) Bibliothèque Jena

Jena est une bibliothèque java pour développer des applications du Web sémantique, elle fournit un environnement de programmation pour RDF, RDF Schema, et OWL. Elle peut être téléchargée à partir du lien : http://jena.sourceforge.net/

g) Editeur d'Ontologies OWL

Néanmoins, *Protégé* n'est pas un outil spécialement dédié à OWL, mais un éditeur hautement extensible, capable de manipuler des formats très divers. Disponible à : http://protege.stanford.edu/

h) Convertisseur WSD2OWL-S

L'outil présenté ***WSDL2OWL-S Converter*** permet de créer une description OWL-S à partir d'une spécification WSDL et il donne à l'utilisateur la possibilité de choisir les noms des paramètres OWL-S ainsi que leur type. Il disponible à http://www.semwebcentral.org/projects/wsdl2owl-s/.

i) OWLS-MX 2.0

C'est un matchmaker sémantique des Web services. Il est disponible à l'adresse http://projects.semwebcentral.org/projects/owls-mx/.

3.1 Exemple d'une ontologie universelle

Dans notre modèle présenté en chapitre 4, nous avons proposé l'utilisation d'une base centrale d'ontologies. Dans ce contexte, plusieurs ontologies ont été proposées couvrant plusieurs domaines d'activités. Certains nombre d'ontologies proposées sont payants, alors que d'autres initiatives sont libres et open source. Un exemple de ces initiatives est celle de **[GAN+06]** qui ont proposé une ontologie universelle dite « concepts.owl » qui perçoit un nombre important des domaines (les plus utilisés). En plus, **[GAN+06]** proposent plus de 240 descriptions OWL-S d'une grandes variété de Web services fournis dans la littérature. Un exemple d'une description OWL-S qui utilise cette ontologie globale est présenté dans l'annexe C.

3.2 Exemple d'une ontologie locale

Dans notre étude de cas, nous avons utilisé une ontologie dite « book.owl » qui contient les concepts et les propriétés décrits dans le listing suivant (les détails de cette ontologie sont présentés dans l'annexe B). Cette ontologie est inspirée de l'ontologie universelle. Elle été développée par l'environnement protégé OWL.

```
<?xml version="1.0"?>
<rdf:RDF
……
<owl:Class rdf:ID="Book">
</owl:Class>
<owl:Class rdf:ID="ISBN">
</owl:Class>
<owl:Class rdf:ID="Price">
</owl:Class>
<owl:Class rdf:ID="Money">
</owl:Class>
<owl:Class rdf:ID="Title">
</owl:Class>
<owl:Class rdf:ID="year">
</owl:Class>
<owl:Class rdf:ID="author">
</owl:Class>
<owl:Class rdf:ID="Edition">
</owl:Class>
```

Classes présentant les différents concepts du domaine.

```
<owl:ObjectProperty rdf:ID="has_ISBN">
<rdfs:domain rdf:resource="#book"/>
<rdfs:range rdf:resource="#ISBN"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="has_Price">
<rdfs:doamin rdf:resource="#book"/>
<rdfs:range rdf:resource="#Price"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="has_Title">
<rdfs:domain rdf:resource="#book"/>
<rdfs:range rdf:resource="#Title"/>
</owl:ObjectProperty>
……
</rdf:RDF>
```

Propriétés

3.3 Implémentation du moteur de recherche local

Pour développer l'algorithme de Matchmaking implémenté dans le moteur de recherche local, nous avons inspiré du travail présenté en **[HAL+03]**. Le rôle de cet algorithme est de

recevoir comme entrée une requête (écrite sous forme d'un fichier OWL), qui contient une description sémantique du Web service désiré, c'est-à-dire la description de ses entrées, ses sorties et son but. L'algorithme la compare avec les descriptions des Web services disponibles et renvoie la liste de ceux qui répondent à cette requête.

La figure 5.2 présente le fonctionnement de ce moteur :

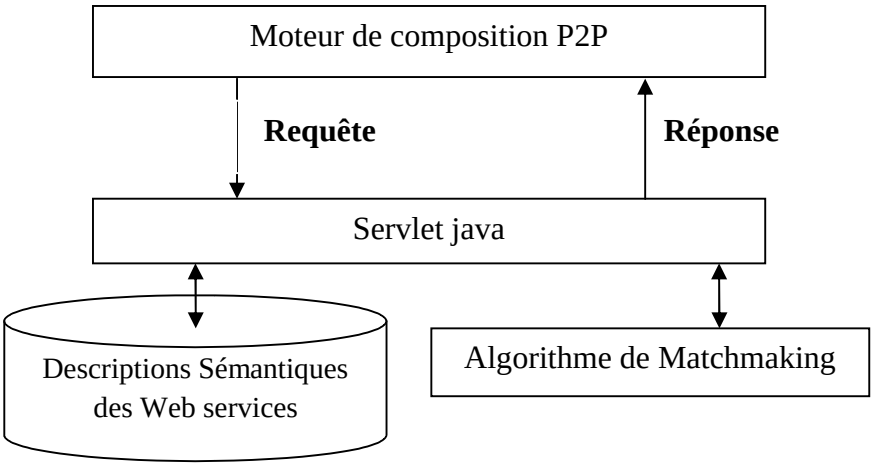


Figure 5.2 : Fonctionnement du moteur de recherche local.

Le moteur de recherche utilise une base de données qui contient quatre tables présentées comme suit : Service_Description, Has_Input, Has_Output et Has_Goal.

Nom du champ	Signification
OWL_URL	Contient l'URL vers le document qui contient la description OWL-S du web service, ce champ est la clé primaire de cette table car il est unique pour chaque Web service.
ONT_URL	Contient l'URL vers l'ontologie qu'utilise ce Web service.
NAME	Le nom du web service.
Contact_Info	Un email ou un site web du propriétaire du Web service.
WSDL_URL	URL du contrat WSDL du web service.

Tableau 5.2 : Structure de la table Service_Description.

Nom du champ	Signification
OWL_URL	URL du document OWL-S du web service.
INPUT_CLASS_URL	URL de la classe qui représente une des entrées du web service

Tableau 5.3 : Structure de la table Has_Input.

Nom du champ	Signification
OWL_URL	URL du document OWL-S du web service.
OUTPUT_CLASS_URL	URL de la classe qui représente une des sorties du web service

Tableau 5.4 : Structure de la table Has_Output.

Nom du champ	Signification
OWL_URL	URL du document OWL-S du web service.
GOAL_CLASS_URL	URL de la classe qui représente le but du web service.

Tableau 5.5 : Structure de la table Has_Goal.

La servlet java permet d'extraire de la base de données les descriptions des Web services disponibles et les descriptions se trouvant dans la requête et les rendre disponible pour l'algorithme de Matchmaking. Ce dernier utilise la liste des descriptions fournie par la servlet, les compare avec la description introduite par la requête de l'utilisateur, et renvoie la liste des Web services qui correspondent à la requête. Cette liste résultante sera communiquée à l'interface utilisateur qui va l'afficher.

Cet algorithme se base sur la comparaison des classes en entrées et en sorties de la requête et les descriptions des Web services disponibles. Les détails algorithmiques sont donnés dans ce qui suit :

Entrée : requête de l'utilisateur écrite en utilisant le langage OWL ;

Sortie : la liste des Web services sémantiques qui répondent à cette requête.

Début

1) Lire la requête en extraire les classes d'entrée et les classes de sortie; /* réalisé par la servlet*/

2) Extraire de la base de données les descriptions des Web services disponibles; /*réalisé par la servlet*/
3) Pour chaque description de Web service extraite faire :

Si

Le nombre des classes en entrée de la description n'est pas égale le nombre de classes en entrée de la requête **ou** le nombre de classes en sortie de la description est différent du nombre de classes en sortie de la requête **ou** le nombre de classes du but de la description n'est pas égale le nombre de classes du but de la requête

Alors

Passer à la description du Web service suivant;

Sinon

Si

(Chaque classe en entrée de la requête a une classe équivalente **ou** supère classe **ou** sous classe dans la description du web service actuel) **ET** (Chaque classe en sortie de la requête a une classe équivalente **ou** supère classe **ou** sous classe dans la description du web service actuel) **ET** (Chaque classe du but de la requête a une classe équivalente **ou** supère classe **ou** sous classe dans la description du Web service actuel)

Alors

Accepter ce Web service ;

Retourner la liste des Web services acceptés ;

Fin

Cet algorithme est implémenté par la méthode"*findcalifiedcondidates*". Les autres méthodes sont définies comme suit (figure 5.3):

- **mySearchEngine :** la classe constructrice
- **readServiceRequest:** lire la requête et extraire les classes d'entrées, de sorties et du but.

Nous avons utilisé la bibliothèque Jena pour automatiser l'inférence sémantique implémentée par ce moteur de recherche. Les détails d'implémentation sont présentés dans l'annexe D.

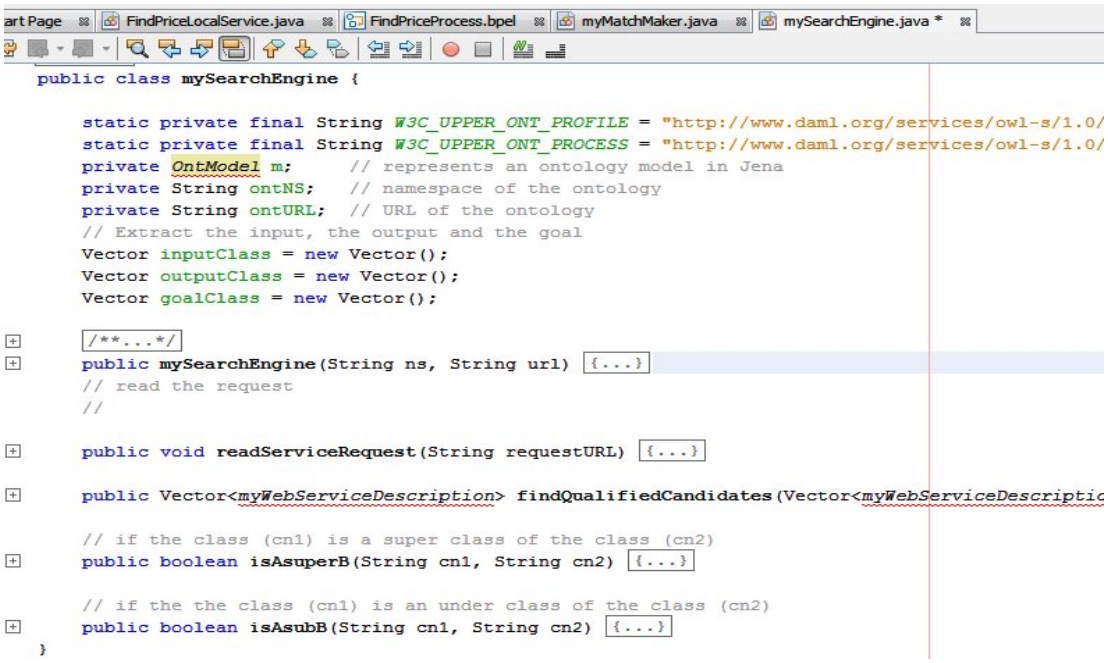


Figure 5.3: Implémentation du moteur de recherche

3.4 Implémentation de l'interface utilisateur

La figure 5.4 présente une portion de l'interface utilisateur issue pour la recherche des livres dans le réseau. L'utilisateur a plusieurs critères en entrée (il pourra soit faire entrer un ou plusieurs). Ces critères sont : le titre du livre, son auteur, sa maison d'édition, l'année de parution. Il devra aussi choisir la monnaie dans laquelle il veut que le prix soit affiché, il a le choix entre le Dollar US (USD) l'Euro (EUR) et le Dinar Algérien (DZD).

Figure 5.6 : Interface de recherche.

3.5 Exemple d'un scenario BPEL

La figure 5.5 présente un autre scenario de l'exemple précédent. Dans ce scénario la composition contient quatre Web services implémentés sur trois nœuds. Le Web service composite est constitué de deux services basiques. Les autres services sont considérés comme étant en réserve. Dans ce scénario, le demandeur a utilisé directement l'ISBN du livre comme paramètre de recherche.

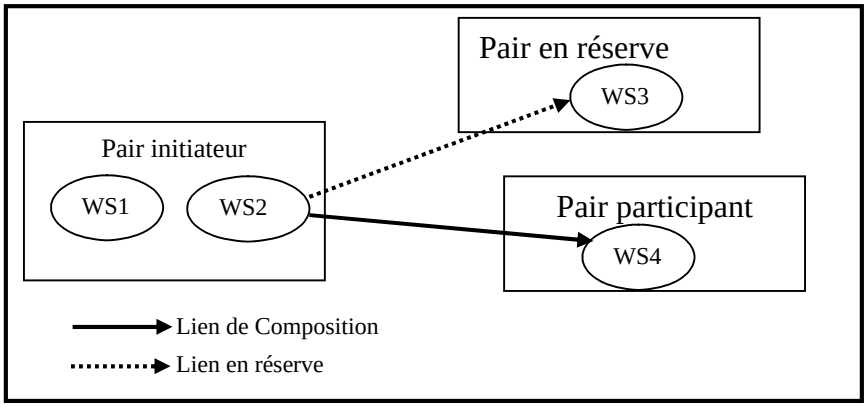


Figure 5.5: Exemple d'une composition.

Les entrées, sorties et les buts des Web services découverts sont présentés dans le tableau suivant :

Web service	Input	Output	But
WS1 (réserve)	Livre info : titre, auteur, Editeur, ISBN	Prix	Affiche le prix en Euro
WS2 (à exécuter)	Livre info : titre, auteur, ISBN,…..	Prix	Affiche le prix en Dollar (US)
WS3 (réserve)	Money en Dollar	Money en Dinar Algérien	Convertisseur USD-AD
WS4 (à exécuter)	Money en Dollar	Money en Dinar Algérien	Convertisseur USD-AD
Composite WS	Livre info (titre, auteur…)	Prix	Affiche le prix en Dinar Algérien

Tableau 5.6: Descriptions des Web services Candidats.

Dans l'étape de composition, le pair initiateur génère un fichier BPEL pour composer le nouveau Web service. Les Web services en réserves seront invoqués si et seulement si les Web services primaires (à exécuter) ne donnent pas un résultat (en panne par exemple).

Le listing suivant montre le code source et la présentation graphique du fichier BPEL généré. Ce dernier prend en considération les quatre Web services candidats pour cette composition (WS1= FindPriceLocalPL, WS2= FindBookPriceServicer, WS3=ReserveConvertor and WS4= OtherPeerConvertorPL). Voir l'annexe E pour plus de détails.

```
<?xml version="1.0" encoding="UTF-8"?>
<process    name="FindPriceProcess"
  ……..
  <import namespace="http://services/"
  location="localhost_8080/FindBookPriceServicerService/
  FindBookPriceServicer.wsdl"
  importType="http://schemas.xmlsoap.org/wsdl/"/>
  ……
<partnerLinks>
    <partnerLink name="FindPriceReservePL" ……..
    partnerLinkType="tns:FindBookPriceServicerLinkType"
    partnerRole="FindBookPriceServicerRole"/>
    <partnerLink name="FindPriceLocalPL" ……
    partnerLinkType="tns:FindPriceLocalServiceLinkType"
    partnerRole="FindPriceLocalServiceRole"/>
    <partnerLink name="OtherPeerConvertorPL" ……
    partnerLinkType="tns:ConvertorServiceLinkType"
    partnerRole="ConvertorServiceRole"/>
    <partnerLink name="ReserveConvertor" ……..
    <partnerLink name="PartnerLink1" ……
    partnerLinkType="tns:FindPriceInDesiredCurrency"
    myRole="FindPriceInDesiredCurrencyPortTypeRole"/>
</partnerLinks>
<variables>……..    </variables>
    <sequence>……..  </sequence>
</process>
```

La figure 5.6 visualise le fichier BPEL d'une manière graphique (générée par l'environnement du développement NetBeans.

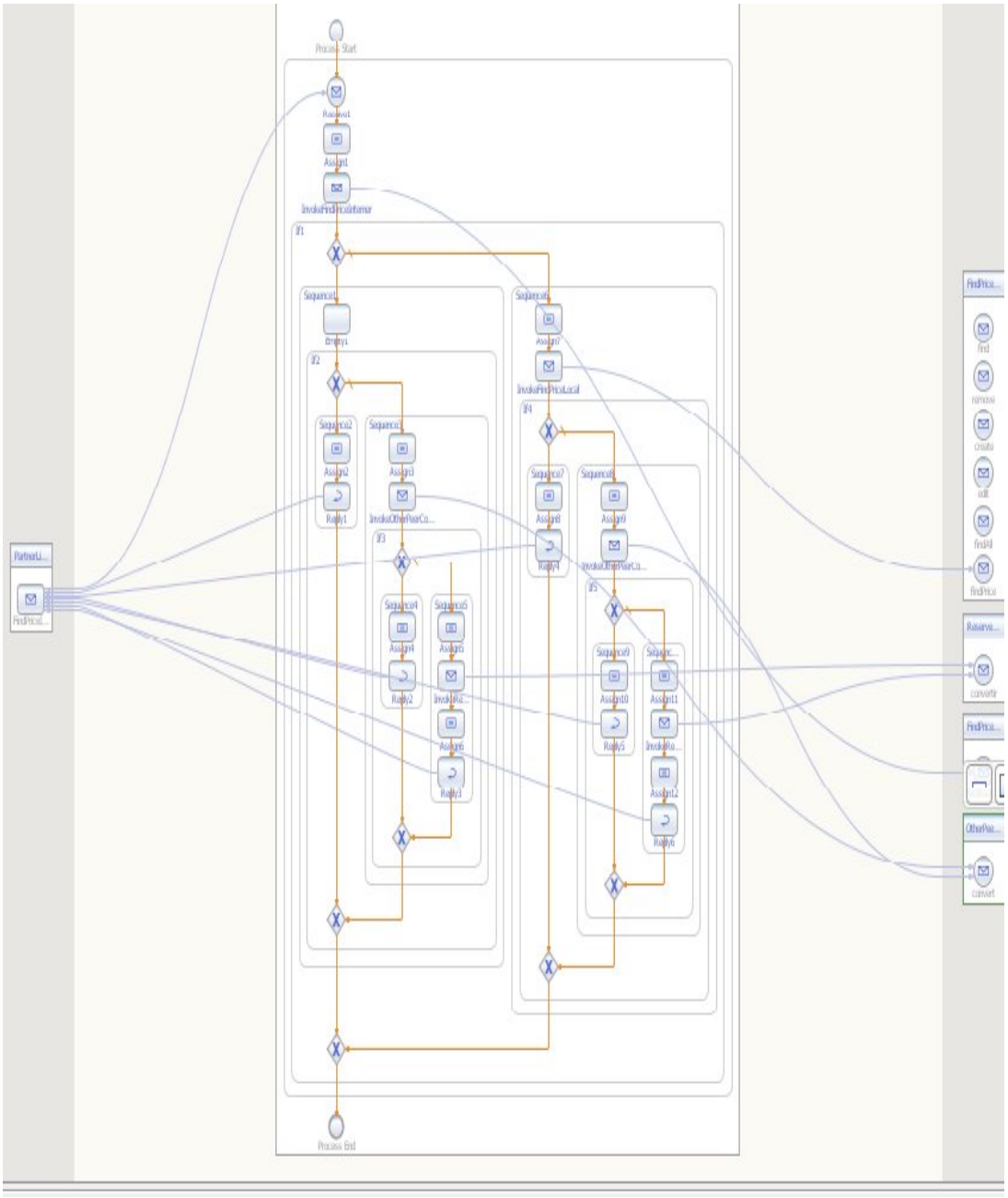


Figure 5.6 : Fichier BPEL correspondant au Web service composite.

4. Implémentation du moteur de composition P2P

L'implémentation du moteur est encore en cours de réalisation. Elle comporte initialement deux parties : la première consiste en l'implémentation de la communication P2P, pour laquelle nous employons la plateforme JXTA ; alors que la deuxième partie présente la simulation de la stratégie proposée dont le but est d'évaluer les performances de notre solution.

JXTA (à prononcer juxta, en provenance du mot juxtapose) est une suite de protocoles Open Source développée par Sun. Son idée est de mettre en place une plate-forme qui faciliterait la construction d'applications Peer-to-Peer **[JXTA]**.

Notre choix s'est porté sur la plateforme JXTA car elle présente plusieurs avantages, à savoir **[GRA02]**:

- **Collaboration :** Elle offre la possibilité à chaque composant d'un réseau de communiquer, de collaborer et de partager des ressources. De plus, elle permet aux pairs participants de s'auto-organiser en groupes.
- **Interopérabilité :** chaque pair peut interagir avec tous les autres pairs même s'ils sont situés derrière un firewall. De plus, JXTA est indépendante des langages de programmation (*C, Java*…), plateformes systèmes (*Windows, UNIX*…), et plates-formes réseaux (*TCP/IP, Bluetooth* …) ; et elle utilise XML comme format de message.
- **Compatibilité avec les technologies des Web services :** à travers l'API **jxta-soap**, JXTA offre la possibilité d'envoyer et de recevoir des messages SOAP en utilisant des pipes JXTA.
- **Open source :** JXTA offre la possibilité d'accéder au code source, de le modifier et même de créer des extensions personnalisées suivant les spécificités de l'application à développée.
- **Ubiquité :** La technologie JXTA fonctionne sur n'importe quel support numérique qu'il s'agisse d'un PC, PDA ou autre.

JXTA offre une suite de protocoles qui nous permet d'implémenter les différentes interactions P2P définies dans notre architecture présentée dans le chapitre précédent. L'annexe F présente un résumé sur les différents concepts définis par JXTA, ses protocoles ainsi que son architecture.

Concernant la communication entre les pairs, JXTA utilise le mécanisme des pipes. Ces derniers permettent aux pairs d'envoyer des messages indépendamment de la typologie du réseau P2P et utilisent la notion de *endpoint* pour désigner les points d'entrée et de sortie d'un canal de communication. JXTA offre deux types de pipes :

- **Unicast :** connectent un peer à un autre pour une communication dans un sens unique. Il existe deux sous classes des pipes unicasts : unidirectionnel non sécurisé et unidirectionnel sécurisé.
- **Propagating :** constitue un tube de sortie à plusieurs entrées de tubes.

Cependant, ces types de pipes sont unidirectionnels et non fiables. Pour cette raison JXTA donne la possibilité de développer des canaux de communication, bidirectionnels et fiables. Cette possibilité est offerte par le package net.jxta.socket implémentant un ensemble de sockets dont les plus importants sont: JxtaSocket, JxtaServerSocket et JxtaMulticastSocket.

Enfin, il est intéressant de signaler que plusieurs sous projets de plateformes JXTA sont encore en cours de développement et que les premiers résultats d'utilisation de JXTA donnent de bons signes quand à l'avenir de cette technologie qui vise à être le nouveau standard pour le développement des applications P2P.

5. Conclusion

Dans la première partie de ce chapitre, nous avons présenté les différents outils techniques utilisés dans cette étape. Ces outils sont classés en deux catégories : technologies liées aux technologies du Web sémantique (Protégé OWL, WSDL2OWL-S, OWL-S MX,…) et celles liées aux systèmes P2P (JXTA).

Dans la seconde partie de ce chapitre, nous avons détaillé l'implémentation des différentes couches de l'architecture ainsi que les composants qui les comportent. Plus précisément, nous avons décris en détaille le moteur de recherche local et le moteur de composition BPEL.

Enfin, il faut mentionner que la phase d'implémentation nécessite une étape d'évaluation de performances de la solution proposée, en utilisant un simulateur des réseaux P2P comme PeerSim ou NS2.

Conclusion générale

Les travaux décrits dans cette thèse portent sur la problématique de découverte et de composition des Web services. Ils ont pour principal objectif de proposer une nouvelle approche pour la découverte des Web services et de concevoir une architecture permettant de concrétiser cette approche.

Il est clairement démontré que la composition des Web services est fortement liée à l'opération de découverte considérée comme essentielle au bon fonctionnement des Web services. A l'origine, le problème de découverte des Web services est natif de deux sous problèmes rencontrés dans l'architecture des Web services, à savoir :

- **La description technique décrite en WSDL** : cette description représente la vitrine du Web service à travers laquelle les clients (d'autres Web services) peuvent comprendre son fonctionnement, les fonctionnalités qu'il fournit, ses modes de communication et d'invocation…etc., bien que la composition dynamique des Web services nécessite une description plus intelligible et plus compréhensible par la machine pour minimiser l'intervention humaine et maximiser la dynamique.
- **Le point central de publication et de recherche** : implémenté généralement à l'aide d'un registre UDDI, le répertoire central de publication et de recherche souffre de plusieurs problèmes. En plus des difficultés classiques des systèmes centralisés, UDDI n'est pas conçu pour les environnements importants comme Internet. Il est plus adapté aux milieux fermés comme l'Intranet et l'Extranet.

Le Web sémantique a été impliqué pour résoudre ces problèmes ; d'une part par la proposition de nouveaux langages de descriptions sémantiques pour les Web services (WSMO, OWL-S, …) et d'autre part par les extensions sémantiques d'UDDI. Un inconvénient est cependant relevé sur ce dernier qui ne permet pas de saisir les relations sémantiques entre les entités et de faire le mapping sémantique.

Pour toutes ces raisons, plusieurs travaux de recherche proposent la décentralisation de la publication et la recherche des Web services à travers la proposition des méthodes distribuées de découverte. C'est dans ce contexte que les systèmes P2P ont été impliqués dans l'enjeu. Ils ont apporté plusieurs solutions aux problèmes rencontrés dans les méthodes centralisées bien que la combinaison du Web sémantique et des systèmes P2P ait montré un certain nombre de limites. Il s'agit notamment de celles liées à la gestion de l'interopérabilité sémantique des Web services

distribués, des types du réseau P2P, des protocoles de communication et de recherche, du manque de méthodologies efficaces pour construire les ontologies ainsi que du manque de maturité des mécanismes de description et de découverte de services. Partant de ce constat, nous nous sommes intéressés à la recherche d'une nouvelle approche de découverte et de composition des Web services basée sur les technologies du Web sémantique et les celles des systèmes P2P.

Dans ce travail, nous avons proposé une approche pour la découverte et la composition des Web services sémantiques dans les systèmes P2P non structurés. Dans cette approche, nous avons défini une stratégie basée sur un algorithme épidémique pour la découverte des Web services. L'objectif de cette étape est de trouver une séquence de services qui peuvent être composés afin de répondre à une requête. Dans cet algorithme, nous avons présentés deux mécanismes pour faire progresser la recherche des Web services dans un réseau P2P non structurés à savoir le chaînage avant et le chaînage arrière.

La deuxième idée proposée dans cette stratégie est l'utilisation de la table de composition qui fournit une méthode fortement distribuée pour découvrir les Web services composés dans un réseau P2P. Cette distribution permet de créer un espace de collaboration où chaque pair peut exploiter les expériences des autres pairs. Cette table se caractérise par la préservation des traces des différentes compositions déjà effectuées avec succès pour une éventuelle future réutilisation.

La table de composition offre plusieurs avantages dans ce contexte. Elle améliore le temps de découverte à travers la réutilisation des Web services déjà composés au lieu de lancer une nouvelle opération de recherche dans le réseau. De plus, la distribution de la table minimise le nombre de pairs « égoïstes » par le fait qu'au fur et à mesure la participation à la composition des nouveaux Web services donne aux pairs participants des tables riches, alors que la distribution permet d'isoler les pairs égoïstes.

Néanmoins, et à cause de la nature volatile des nœuds dans un réseau P2P, la distribution de la table de composition pose un problème de cohérence des données contenues dans cette table. Plusieurs compositions peuvent devenir inactives lorsqu'un pair participant quitte le réseau. Pour cette raison, nous avons enrichi la stratégie initialement proposée par des algorithmes de notification afin d'assurer la cohérence des données des tables des différents pairs. De plus, nous avons proposé un algorithme qui permet de réparer les compositions interrompues à cause de l'absence d'un pair participant.

La deuxième partie de notre travail présente une architecture qui implémente la stratégie proposée dans la première étape. L'architecture est composée d'une base centrale d'ontologies et d'un framework distribué sur les différents pairs. L'objectif de cette base est d'assurer l'interopérabilité et la compatibilité des descriptions sémantiques relatives aux différents Web services. Ce framework implémente les différents algorithmes de découverte, de notification et de réparation. Il contient trois couches principales permettant de gérer les communications externes : avec les autres pairs, les compositions locales de chaque pair et la gestion des descriptions sémantiques. Pour cette dernière tâche nous avons choisi l'utilisation de langages sémantiques matures qui sont OWL et OWL-S.

Enfin, nous avons présenté un exemple qui motive notre approche et nous avons décrit le fonctionnement des différents composants du framework présenté dans l'étape précédente.

Les travaux proposés dans cette thèse ouvrent plusieurs perspectives à court et à long terme.

Dans notre travail, nous avons pris en considération les problèmes de la sémantique, de la découverte et de la composition dynamique des Web services. Néanmoins, notre approche nécessite plus d'améliorations en termes de critères de qualité de service. L'enrichissement de l'algorithme de découverte par des critères de sélection permettrait d'améliorer le résultat de la recherche. De plus, il serait intéressant de rajouter un mécanisme de représentation de la qualité de service au niveau de la requête de découverte.

Une autre perspective à court terme est celle de l'évaluation des performances des différents algorithmes proposés dans notre approche. A cause de la difficulté de l'implémentation de cette solution dans un réseau à grande échelle, une étape de simulation est primordiale pour présenter ce type de réseau. Pour cette raison, nous sommes en cours d'étudier l'utilisation de simulateurs des réseaux P2P comme NS2 ou PeerSim, afin d'implémenter et simuler le fonctionnement de notre framework dans un réseau P2P non structuré. Par conséquent, nous pourrons comparer plusieurs protocoles non structurés (par inondation, Gnutella v0.4, etc.).

L'étape de simulation et d'évaluation de performances nous permettra de bien définir les critères de base qui influent sur le temps de recherche et la qualité du résultat retourné. Voici une liste non exhaustive des propriétés que nous voulons observer :

- Le nombre total des pairs interrogés et des pairs pertinents

- Le nombre des requêtes propagées, des résultats retournés ainsi que le nombre des messages nécessaires pour le fonctionnement du réseau.
- Le TTL (Time To Live) adéquat pour retourner un résultat.
- La taille adéquate de la table de composition.

Comme perspectives à long terme, nous pouvons enrichir les algorithmes proposés avec une approche probabiliste afin de filtrer les pairs pertinents du réseau. De plus, nous voulons étudier la possibilité de regrouper des pairs (clusters) suivant le contenu de la table de composition de chaque pair. Cette proposition peut résoudre quelques problèmes rencontrés dans les systèmes P2P non structurés ; particulièrement du passage à l'échelle. Dans cette nouvelle vision, chaque groupe est géré par un Super-Pair implémentant une table de composition commune.

Références Bibliographiques

Références Bibliographiques

[ALE+06] D. Alexandre, E. Samuel, "*Peer-To-Peer*", Université Claude Bernard LYON 1, 2006.

[AND+04] S. Androutsellis-Theotokis and D. Spinellis, "A Survey of Peer-to-Peer Content Distribution Technologies", ACM Computing Surveys, Vol. 36, No. 4, pp. 335–371, December 2004.

[ANT+02] G. Antoniou, L. Bougé, T. Priol, "Partage de mémoire à très grande échelle sur des réseaux pair-à-pair", Institut National De Recherche En Informatique Et En Automatique INRIA, Mars 2002. ftp://ftp.inria.fr/INRIA/publication/publi-pdf/RR/RR-4410.pdf

[ANT+O8] G. Antoniou et F. van Harmelen "*A Semantic Web Primer*", The MIT Press Cambridge, Massachusetts London, England, Cooperative information systems, 2008.

[ARA+06]: K. Arabshian and H. Schulzrinne, *"An Ontology-based Hierarchical Peer-to-Peer Global Service Discovery System"*, Journal of Ubiquitous Computing and Intelligence (JUCI), volume 1 (2), pp. 133-144, 2006.

[BAN+04]: F. Banaei-Kashani, C. Chen, and C. Shahabi, *"WSPDS: Web Services Peer-to-Peer Discovery Service"*, International Conference on Internet Computing (IC'04), 733–743, Las Vegas, Nevada, USA, 2004.

[**BAT+10]**:S. Batra, S. Bawa "Review of Machine Learning Approaches to Semantic Web Service Discovery", Journal of Advances in Information Technology (JAIT) vol. 1, no. 3, August 2010.

[BEL09] : Y. BELAID, "*SERVICE WEB – SOAP*", Centre d'enseignements de Grenoble, 2009.

[BEN06] : A. Benoit, "*Algorithmique des réseaux et des télécoms, Chapitre 2 : Réseaux Pair à Pair*". ENS Lyon, 2006.

[BEN+09]: D. Benmerzoug, M. Gharzouli and M. Boufaida "*Formalisation and Verification of Web Services Composition based on BPEL4WS*". In: First Workshop of Web services in Information Systems (WWS'09), pp. 37-47, Algies, Algeria, 2009.

[BER+05] : D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella, "Automatic *Composition of Transition-based Semantic Web Services with Messaging*", In Proc. of VLDB, Trondheim, Norway, 2005, pp. 613-624.

[BER+01]: T. Berners-Lee, J. Hendler, et O. Lassila, "*The Semantic Web*". Scientific Amercian, 2001.

[BIA+06]: D. Bianchini, V.D Antonellis, M. Melchiori, D. Salvi, *"Peer-to-Peer Semantic-based Web Service Discovery: State of the Art"*, ESTEEM: Emergent Semantics and cooperaTion in multi-knowledgE EnvironMents, 2006. Disponible à www.dis.uniroma1.it/esteem/docs/deliverable%20soa/SoA_UNIBS.pdf

[BOU09] : K. Boukadi, "*Coopération interentreprises à la demande:Une approche flexible à base de services adaptables*", thèse de doctorat, l'École Nationale Supérieure des Mines de Saint-Étienne, 2009.

[Bro96] : Alan W Brown. *Component-based software engineering : selected papers from the Software Engineering Institute*. Los Alamitos, CA: IEEE Computer Society Press, 1996.

[BRU+05]: J. Bruijn, H. Lausen, R. Krummenacher, A. Polleres, M. Kifer et D. Fensel, "*Deliverable D16.1v0.2, The Web Service Modeling Language WSML*". Final Draft, *WSMO project*, http://www.wsmo.org/TR/d16/d16.1/v0.2/20050320/, 2005.

[BUS+04] : C. Bussler, et al., "*WSMO"*. The Eleventh International Conference on Artificial Intelligence: Methodology, Systems, 2004.

[BOU+07] : J. Bourdon, P. Beaune, H. Fiorino, "*Architecture multi-agents pour la composition automatique de Web services*", Atelier IAWI - Plate-forme AFIA, 2007.

[BRA+09]:M. Brahimi, L. Seinturier, M. Boufaïda: "*Multi-Agent Architecture for Developing Cooperative E-Business Applications*", International Journal of Information Systems and Supply Chain Management (IJISSCM), volume 2, N°4: pp.43-62, 2009.

[CAR06]: J. Cardoso, "*Approaches to Developing Semantic Web Services*", International Journal of Computer Science, 1 (1), 2006.

[CAS+02]: F. Casati, et M.-C Shan, *"Event-Based Interaction Management for Composite Eservices in eFlow"*, Information Systems Frontiers, 4(1), pp. 19-31, 2002.

[CHA02] : J-M Chauvet «*Services Web avec SOAP, WSDL, UDDI, ebXML…*» Edition EYROLLES, 2002.

[**CHR+01] :** E. Christensen, F. Curbera, G. Meredith, S. Weerawarana « *Web Service Description Language (WSDL) 1.1* », spécification W3C, 2001.

[CHR+08]: B. **A.** Christudas, M. Barai, V. Caselli, "*Service Oriented Architecture with Java, Using SOA and web services to build powerful Java applications*", Edition Packet Publishing, 2008.

[CLA+02] I. Clarke, S. G. Miller, T.W. Hong, O. Sandberg, B. Wiley "Protecting free expression online with freenet". IEEE Internet Computing, 6(1), pp. 40–49, 2002.

[CRO05] : Antoine CROCHET-DAMAIS, JDN Solution, "*Comprendre les Architecture Orientée Services (SOA)*", 2005. http://www.journaldunet.com/solutions/dossiers/pratique/soa

[EME+04]: F. Emekci, O.D. Sahin, D. Agrawal, and A. El Abbadi, "A Peer-to-Peer Framework for Web Service Discovery with Ranking", *In the Proc of the IEEE International Conference on Web Services (ICWS'04)*, California USA, 2004, pp. 192-199.

[ESS+05]: T. ESSAFI, N. DORTA and D. SERET, "A Scalable Peer-to-Peer Approach To Service Discovery Using Ontology", *In the Proc of 9th World Multiconference on Systemics, Cybernetics and Informatics.* Orlando, 2005.

[FEL+03] : G. Feltin, G. Doyen et O. Festor, "Les protocoles Peer-to-Peer, leur utilisation et leur détection", LORIA - Campus Scientifique, Octobre 2003.

[FEN+02 a]: D. Fensel, C. Bussler, A. Maedche: Semantic Web Enabled Web Services. International Semantic Web Conference, 2002.

[FEN+02 b] D. Fensel et C. Bussler, "*The Web Service Modeling Framework WSMF*". In Electronic Commerce Research and Applications, 1(2), *Elsevier, Scinces B. V.*, 2002.

[FOU+06] : X. Fournier-Morel, P. Grojean, G. Plouin, C. Rognon, "SOA le guide de l'architecte ", Edition Dunod, 2006.

[GAN+06]: Y. Ganjisaffar and H. Saboohi, "Semantic Web Central: Project sws-tc", 2006, http://projects.semwebcentral.org/projects/sws-tc/

[GHA04] : M. Gharzouli "Proposition d'un processus de développement des portails d'entreprises basé sur les services Web", Mémoire de magistère, Université Larbi Tebessi de Tébessa, novembre 2004.

[GHA+11]: M. Gharzouli and M. Boufaida "PM4SWS: A P2P Model for Semantic Web Services Discovery and Composition". Journal of Advances in Information Technology, Special Issue on Advances in P2P Technology, Vol 2-N°1, Academy Publisher, (February 2011).

[GHA+10]: M. Gharzouli and M. Boufaida "A Distributed P2P-based architecture for semantic Web services discovery and composition". In: 10th Annual International Conference on New Technologies of Distributed Systems (NOTERE), pp. 315-320. IEEE, Tozeur, Tunisia, 2010.

[GHA+09]: Mohamed Gharzouli & Mahmoud Boufaida "A Generic P2P Collaborative Strategy for Discovering and Composing Semantic Web Services". In: Fourth International Conference on Internet and Web applications and Services (ICIW), pp. 449-454. Venice/ Mestre, Italy (2009).

[GHA+08] : M. Gharzouli and M. Boufaida "Préservation des données pour la composition des Web services sémantiques dans les systèmes Pair-à-Pair non structurés", Workshop sur la Cohérence des Données en Univers Réparti CDUR'08, 23-27 Juin, 2008, Lyon, France.

[GRA02]: J.D Gradecki "*Mastering JXTA: Building Java Peer-to-Peer Applications*", Edition: John Wiley & Sons, 2002.

[GRU93]: T. R. Gruber "A translation approach to portable ontologies", *Knowledge Acquisition*, 5(2):199-220, 1993.

[HAA+04]: P. Haase, R. Siebes, F. v.Harmelen, "Peer Selection in Peer-to-Peer Networks with Semantic Topologies", *In the Proc of ICSNW*, Paris, France, 2004, pp. 108-125.

[HAL+03] M. Hall and L. Brown, " Core Servlets and JavaServer Pages" Prentice Hall PTR, 2003

[HAM+03] S. Hammoudi et D. Lopes, "*Introduction aux Services Web*", 2003 http://www.dee.ufma.br/~dlopes/course/WebServices/presentation-WS-3.3_economic.pdf

[HU+05] J. Hu, C. Guo, H. Wang, and P. Zou, "Web Services Peer-to-Peer Discovery Service for Automated Web Service Composition", *Springer-Verlag Berlin Heidelberg (LNCS 3619), ICCNMC 2005*, Zhangjiajie China, 2005, pp. 509-518.

[IZZ06] : Saïd IZZA, " *Intégration des Systèmes d'Information* : *Une approche flexible basée sur les services sémantiques*", Thèse de doctorat, l'École Nationale Supérieure des Mines de Saint-Étienne, 2009.

[JIN+05]:H. Jin, H. Wu, Y. Li, and H. Chen, "An Approach for Service Discovery based on Semantic Peer-to-Peer", *In the Proc of ASIAN*, Kunming, China, 2005, pp. 259-260.

[JUR+06]: M. B. Juric, B. Mathew, P. Sarang, "*Business Process Execution Language for Web Services"*, 2eme Edition Packet Publishing, 2006.

[KEL+06] : P. Kellert, et F. Toumani, "*Les services web sémantiques*", Revue I3 (Information-Interaction-Intelligence), 2006.

[KUB+00]: J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, , R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao, "*Oceanstore : An architecture for global-scale persistent storage*", In Proceedings of ACM ASPLOS, 2000.

[KUN+06]: P. Kungas and M. Matskin, "Semantic Web Service Composition Through a P2P-Based Multi-agent Environment", *Springer-Verlag Berlin Heidelberg (LNAI 4118), AP2PC 2005*, Utrecht The Netherlands, 2006, pp. 106-119.

[LAU+03] : P. Laublet, C. Reynaud, J. Charlet "*Sur quelques aspects du Web sémantique*", Actes des deuxièmes assises nationales du GdR I3, 2003.

[LIU+05]:J. Liu, N. Gu, Y. Zong, Z. Ding, S. Zhang and Q. Zhang, **"**Web Services Automatic Composition Based on QoS", *IEEE Computer Society, In the Proc of the 2005 IEEE International Conference on e-Business Engineering (ICEBE'05)*, Beijing China, 2005, pp. 607-610.

[LOO06] A. W. S. Loo, “Peer-to-Peer Computing: Building Supercomputers with Web Technologies», Edition Springer, 2006.

[LV+02]: Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker “Search and Replication in Unstructured Peer-to-Peer Networks”, *Proc of the 16th international conference on Supercomputing*, New York, USA, 2002, pp. 84-95.

[MAN+07]: F. Mandreoli, A. M. Perdichizzi, and W. Penzo, “A P2P-based Architecture for Semantic Web Service Automatic Composition”, *IEEE computer Society DOI 10.1109/DEXA*2007, Regensburg Germany, 2007, pp. 429-433.

[MIL+03] : D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins and Z. Xu, “Peer-to-Peer Computing”, HP Laboratories Palo Alto, juillet 2003, http://www.hpl.hp.com/techreports/2002/HPL-2002-57R1.pdf

[MEL04] : T. MELLITI “ *Interopérabilité des services Web complexes. Application aux systèmes multi-agents*“, thèse de doctorat, Université Paris IX Dauphine 2004.

[MUT+02]: A. Muthitacharoen, R. Morris, T.M. Gil, B. Chen, “*Ivy : A read/write peer to peer file system”*. In Proceedings of 5th Symposium on Operating Systems Design and Implementation, 2002.

[MOD02]: T. Modi « *WSIL: Do we need another Web Services Specification? Explaining the difference between UDDI* », 2002.

[NAT+94]: Y. V. Natis, R. W. Schulte “Introduction to Service Oriented Architectue », Gartner, 2003.

[ORL+03]: M. E. Orlowska, S. Weerawarana, M. P. Papazoglou, J. Yang: “Service-Oriented Computing”, ICSOC: First International Conference on Services Oriented Computing, Trento, Italy, 2003.

[PAP+07]: M. P. Papazoglou, et W. J.van den Heuvel W. J., "Service Oriented Architectures: Approaches, Technologies, and Research Issues". *VLDB Journal*, Volume 16, N° 3, pp. 389-415, 2007.

[PAO+02]: M. Paolucci, T. Kawamura, T.R. Payne and K. Sycara, **"**Semantic Matching of Web Services Capabilities", *in the Proc of the First International Semantic Web Conference (ISWC),* Sardinia, Italy, , 2002, pp. 333-347.

[PAO+03]: M. Paolucci, K. Sycara, T. Nishimura and N. Srinivasan**,** "Using DAML-S for P2P Discovery", *In the Proc of International Conference on Web Services (ICWS).* Las Vegas, Nevada, USA, 2003, pp. 203-207.

[POT+03] S. Potts et M. Kopak. *"Teach yourself web services in 24 hours"*, Edition Sams Publishing, 2003.

[PFI+96] : C. Pfister et C. Szyperski. Why objects are not enough. In *Proceedings, International Component Users Conference,* Munich, Germany, 1996.

[RAG08]: K. Rageb "An Autonomic <K, D>-Interleaving Registry Overlay Network for Efficient Ubiquities Web Services Discovery Service", Journal Information Processing Systems (JIPS) Vol. 4, no.2, June 2008.

[RAT+01]: S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content addressable network", In *Proc of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications,* ACM SIGCOMM, California, USA, 2001, pp. 161-172

[ROS+08]:M. Rosen, B. Lublinsky, K. T. Smith, M. J. Balcer, "Applied *SOA Service-Oriented Architecture and Design Strategies"* Edition Wiley Publishing, 2008.

[ROW+01]: A. Rowstron and P. Druschel, "*Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems*", *In the Proc of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware* 2001), Heidelberg, Germany, 2001.

[SAH+05] O. D. Sahin, C. E. Gerede, D. Agrawal, A. El Abbadi, O. Ibarra, and J. Su, "SPiDeR: P2P-Based Web Service Discovery", *In the Proc of Third International Conference on Service-Oriented Computing (ICSOC),* Amsterdam, The Netherlands, 2005, pp. 157-169.

[SAR+04] N. Sarshar, P.O Boykin V.P Roychowdhury *"Percolation Search in Power Law Networks: Making Unstructured Peer-To-Peer Networks Scalable"*, The 4th IEEE International Conference on Peer-to-Peer Computing, Zurich, Switzerland, 2004.

[SCH+04]:A. Schmidt, C. Winterhalter "*User Context Aware Delivery of E-Learning Material: Approach and Architecture*", Journal of Universal Computer Science (JUCS) vol.10, no.1, January 2004

[SEG+04]A.E.F Seghrouchni, S. Haddad, T. Melitti, A. Suna, "*Interopérabilité des systèmes multi-agents à l'aide des services Web* ", les Journées Francophones sur les Systèmes Multi-Agents, Paris, France, pp. 91-104, 2004.

[SHI00]: C. Shirky "Clay Shirky on P2P", Novembre 2000, http://www.scripting.com/Davenet/2000/11/15/clayShirkyOnP2p.html

[STE02]: M. Stevens **"***Service-Oriented Architecture Introduction*", 2002. http://www.developer.com/services/article.php/1010451

[STE09]: M. Stevens, "*SOA*" November 2009,

[STO+01]: I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications", *In Proceedings of the ACM SIGCOM*, California, USA, 2001, pp.149-160.

[Soap]: Simple Object Access Protocol (SOAP) 1.1, spécification W3C, 2000.

[Szy97] : Clemens Szyperski "*Component software : beyond object-oriented programming"*. New York : ACM Press Harlow, England Reading, Mass : Addison-Wesley, 1997.

[TAM06] : R.B Tamrout **"**JREGISTRE: Un Registre UDDI Extensible", Mémoire de la maitrise en informatique, Université du Québec à Montréal, CANADA, 2006.

[TAN+06]: K-L. Tan, C-S. Lee, and H-N Chua "*Models to Customise Web Service Discovery Result using Static and Dynamic Parameters*", Proceedings of world academy of sciences engineering, Volume 12, 2006.

[TAN+07] : X. Tang, C. Jiang, Z. Ding, "Automatic Web Service Composition Based on Logical Inference of Horn Clauses in Petri Net Models", *In Proc of IEEE International Conference on Web Services (ICWS)*, Utah, USA, 2007, pp. 1162-1163.

[VAD+11]:G. Vadivelou, E.IIavarasan, S. Prasanna, *"Algorithm for Web Service Composition using Multi-Agents*", International Journal of Computer Applications (0975 – 8887),Volume 13– No.8, January 2011.

[VER+05] : K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller, *"METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services"*, Journal of Information Technology and Management, Special Issue on Universal Global Integration, 6(1):17–39, 2005.

[VEZ05] : A. VEZAIN, "*LES SERVICES WEB – Présentation générale*", Version 0.0.1, Association HE.R.M.E.S, Château du Montet, 2005.

[VU+05]: L-H. Vu, M. Hauswirth and K. Aberer, "Towards P2P-based Semantic Web Service Discovery with QoS Support", *In the Proc of Business Process Management Workshops*, Nancy France, 2005, pp.18-31.

[WAN+07]: C. Wang, J. Lu, and G. Zhang, "Generation and Matching of Ontology Data for the Semantic Web in a Peer-to-Peer Framework", *Springer-Verlag Berlin Heidelberg*, *Proc of APWeb/WAIM*. LNCS 4505, Huang Shan, china, 2007, pp. 136–143.

[WOO75]: W. A. Woods, "*What's in a link: foundations of semantic networks"*. In D.G. Bobrow & A. M. Collins, 1975.

[YU07] : L. Yu "*Introduction to semantic web and semantic web services* ", Chapman & Hall/CRC, 2007.

[ZHE+09]: Z. Zhengdong, H. Yahong, L. Ronggui, W. Weiguo, L. Zengzhi, "*A P2P-Based Semantic Web Services Composition Architecture*". In: IEEE International Conference on e-Business Engineering, pp.403-408, Macau, China (2009).

Logiciels, protocoles et plateformes P2P

[BitTorrent] : disponible sur http://www.bittorrent.com/intl/fr/

[eDonkey]: eDonkey, (edonkey 2000), http://www.edonkey2000.com/

[GNUv04]:The annotated Gnutella protocol specification v0.4, disponible à: http://rfcgnutella.sourceforge.net/developer/stable/index.html

[Gnutella] : Gnutella community, http://www.gnutella.com/

[Groove]: Groove, http://www.groove.net

[ICQ] : ICQ, http://web.icq.com/.

[INFS] : InfraSearch, http://www.oreillynet.com/pub/d/234

[Jabber]: Foundation, J.S.: Jabber, http://www.jabber.org/.

[JXTA] : Site officiel de JXTA, https://**jxta**.dev.java.net/

[Kazza] : Kazaa media desktop, www.kazaa.com

[Magi] : http://www.magicsoftware.com/en/

[MSN] : Microsoft Msn messenger, http://messenger.fr.msn.be/

[Napster]: The ultimate digital music service *"Napster"*, www.napster.com

[NextPage] : http://www.nextpage.com/

[OpenChord] : Open Chord, une implémentation open source de Chord, disponible à http://sourceforge.net/projects/open-chord/

[PPD]: Power Point Distribué, powerpoint.lesiteofficiel.net

[SETI]: Seti@home, http://setiathome.berkeley.edu/.

[.NET]: Microsoft asp.NET, www.asp.net/FR

Spécifications, standards et logiciels des Web services et du Web sémantique

[DAML]: Darpa Agent Markup Language, http://www.daml.org/

[ebXML] : e-business XML, spécification d'OASIS, disponible à http://www.ebxml.org/

[finder]: Service finder: a search engine for web services discovery, disponible à www.service-finder.eu

[METEOR-S]: Semantic Web services and Processes, disponible à http://lsdis.cs.uga.edu/projects/meteor-s/

[OWL]: Web Ontology Language, Standard W3C, http://www.w3.org/TR/owl-features/

[OWL-S] : OWL-S: "Semantic Markup for Web Services", spécification W3C, http://www.w3.org/Submission/OWL-S/

[owls-editor] : éditeur pour lire, éditer et écrire des ontologies OWL, disponible à http://projects.semwebcentral.org/projects/owlseditor/

[owls-mx]: Hybrid OWL-S Web Service Matchmaker, http://projects.semwebcentral.org/projects/owls-mx/ or http://www-ags.dfki.uni-sb.de/~klusch/owls-mx/

[protégé] : Protégé OWL, disponible à http://protege.stanford.edu/plugins/owl/

[RDF]: Ressource Description Framework, Standard W3C, http://www.w3.org/RDF/

[WSDL-S]: WSDL-S, "*Web Service Semantics - WSDL-S, Version 1.0*". W3C Member Submission, http://www.w3.org/Submission/WSDL-S/, 2005.

[wsdl2owls]: Semantic Web Central: Project wsdl2owl-s, "translator from wsdl to owl-s", 2004, http://projects.semwebcentral.org/projects/wsdl2owl-s/

[WSIL] : Web Services Inspection Language, spécification contribuée par IBM et Microsoft, disponible à http://www.ibm.com/developerworks/library/specification/ws-wsilspec/

[WSMO]: "Web Service Modeling Ontology Project Deliverables". *WSMO Project*, 2005.

Annexes

Annexe A

Le listing suivant présent l'écriture de la requête de l'utilisateur en utilisant le modèle OWL-S.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE uridef[
   <!ENTITY rdf       "http://www.w3.org/1999/02/22-rdf-syntax-ns">
   <!ENTITY rdfs      "http://www.w3.org/2000/01/rdf-schema">
   <!ENTITY owl       "http://www.w3.org/2002/07/owl">
   <!ENTITY service   "http://www.daml.org/services/owl-s/1.0/Service.owl">
   <!ENTITY process   "http://www.daml.org/services/owl-s/1.0/Process.owl">
   <!ENTITY profile   "http://www.daml.org/services/owl-s/1.0/Profile.owl">

   <!ENTITY domainOnt "file://localhost/C:\\Documents and
Settings\ontologies\\book.owl">
   ]>
<rdf:RDF
   xmlns:rdf=       "&rdf;#"
   xmlns:rdfs=      "&rdfs;#"
   xmlns:owl=       "&owl;#"
   xmlns:service=   "&service;#"
   xmlns:process=   "&process;#"
   xmlns:profile=   "&profile;#"
   xmlns:domainOnt= "&domainOnt;#"
   xmlns=           "&DEFAULT;#">

   <owl:Ontology about="">
      <owl:imports rdf:resource="&rdf;" />
      <owl:imports rdf:resource="&rdfs;" />
      <owl:imports rdf:resource="&owl;" />
      <owl:imports rdf:resource="&service;" />
      <owl:imports rdf:resource="&profile;" />
      <owl:imports rdf:resource="&process;" />
      <owl:imports rdf:resource="&domainOnt;" />
   </owl:Ontology>

<profile:Profile rdf:ID="getBookPrice">
      <!-- Descriptions of the parameters that will be used by IOPEs -->
      <profile:hasInput rdf:resource="#input1-bookInfo"/>
      <profile:hasOutput rdf:resource="#output1-bookPrice"/>
      <profile:hasOutput rdf:resource="#goal-*input1-BookInfo*output1-
book:Price//Money:EUR"/>
</profile:Profile>

   <process:Input rdf:ID="input1-bookInfo">
   <process:parameterType rdf:resource="&domainOnt;#Edition"/>
   </process:Input>

   <process:UnConditionalOutput rdf:ID="output1-bookPrice">
      <process:parameterType rdf:resource="&domainOnt;#Price"/>
   </process:UnConditionalOutput>

   <process:UnConditionalOutput rdf:ID=" goal-*input1-BookInfo*output1-
book:Price//Money:EUR ">
      <process:parameterType rdf:resource="&domainOnt;#Edition#Price"/>
   </process:UnConditionalOutput>

</rdf:RDF>
```

Annexes B

Une vue simplifiée de l'ontologie book.owl.

```
<?xml version="1.0"?>
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xml:base="file://localhost/C:\\Documents and
  Settings\ontologies\\book.owl">
<owl:Ontology rdf:about="file://localhost/C:\\Documents and
  Settings\ontologies\\book.owl">
    <rdfs:comment>An OWL ontology that describes a
  book</rdfs:comment>
    <rdfs:label>book ontology</rdfs:label>
    <owl:versionInfo>book.owl 1.0</owl:versionInfo>

</owl:Ontology>
<owl:Class rdf:ID="Book">

</owl:Class>

<owl:Class rdf:ID="ISBN">

</owl:Class>

<owl:Class rdf:ID="Price">

</owl:Class>

<owl:Class rdf:ID="Money">

</owl:Class>

<owl:Class rdf:ID="Title">

</owl:Class>

<owl:Class rdf:ID="Year">

</owl:Class>

<owl:Class rdf:ID="Author">

</owl:Class>

<owl:Class rdf:ID="Edition">

</owl:Class>
```

Classes présentant les différents concepts du domaine.

```
<owl:ObjectProperty rdf:ID="has_ISBN">
<rdfs:domain rdf:resource="#Book"/>
<rdfs:range rdf:resource="#ISBN"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="has_Price">
<rdfs:doamin rdf:resource="#Book"/>
<rdfs:range rdf:resource="#Price"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="has_Title">
<rdfs:domain rdf:resource="#Book"/>
<rdfs:range rdf:resource="#Title"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="has_Year">
<rdfs:domain rdf:resource="#Book"/>
<rdfs:range rdf:resource="#Year"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="has_Author">
<rdfs:domain rdf:resource="#Book"/>
<rdfs:range rdf:resource="#Author"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="has_Edition">
<rdfs:domain rdf:resource="#Book"/>
<rdfs:range rdf:resource="#Edition"/>
</owl:ObjectProperty>
</rdf:RDF>
```

Propriétés

Annexes C

Le listing suivant présente une des 240 descriptions OWL-S fournies par la base ontologique globale. Cette description OWL-S est basée sur l'ontologie globale nommée « concepts.owl ». Elle décrite un Web service de vente des livres.

```
<?xml version='1.0' encoding='UTF-8'?>
<rdf:RDF
 xmlns:owl = "http://www.w3.org/2002/07/owl#"
 xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
 xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:service = "http://www.daml.org/services/owl-
s/1.1/Service.owl#"
 xmlns:process = "http://www.daml.org/services/owl-
s/1.1/Process.owl#"
 xmlns:profile = "http://www.daml.org/services/owl-
s/1.1/Profile.owl#"
 xmlns:grounding = "http://www.daml.org/services/owl-
s/1.1/Grounding.owl#"
 xmlns:expr = "http://www.daml.org/services/owl-
s/1.1/generic/Expression.owl#"
 xmlns:swrl = "http://www.w3.org/2003/11/swrl#"
 xml:base = "http://127.0.0.1/services/BookPrice.owl">

<owl:Ontology rdf:about="">
    <owl:imports
rdf:resource="http://127.0.0.1/ontology/Concepts.owl"/>
</owl:Ontology>

<service:Service rdf:ID="BookPrice">
    <service:presents rdf:resource="#BookPrice-Profile"/>
    <service:describedBy rdf:resource="#BookPrice-Process-Model"/>
    <service:supports rdf:resource="#BookPrice-Grounding"/>
</service:Service>

<profile:Profile rdf:ID="BookPrice-Profile">
    <service:isPresentedBy rdf:resource="#BookPrice-Service"/>
        <profile:serviceName xml:lang="en">
           Book Price
    </profile:serviceName>
    <profile:textDescription xml:lang="en">
           Returns the price of a book if it is available in the
stock.
    </profile:textDescription>
    <profile:hasInput rdf:resource="#Book"/>
    <profile:hasOutput rdf:resource="#PriceOfBook"/>
    <profile:hasPrecondition rdf:resource="#BookInStock"/>
</profile:Profile>
```

```
<process:ProcessModel rdf:ID="BookPrice-Process-Model">
     <service:describes rdf:resource="#BookPrice-Service"/>
     <process:hasProcess rdf:resource="#BookPrice-Process"/>
</process:ProcessModel>
<process:AtomicProcess rdf:ID="BookPrice-Process">
   <process:hasInput rdf:resource="#Book"/>
   <process:hasOutput rdf:resource="#PriceOfBook"/>
   <process:hasPrecondition>
      <expr:SWRL-Condition rdf:ID="BookInStock">
          <rdfs:comment>Is Book in stock</rdfs:comment>
          <expr:expressionLanguage
rdf:resource="http://www.daml.org/services/owl-
s/1.1/generic/Expression.owl#SWRL" />
          <expr:expressionBody rdf:parseType="Literal">
              <swrl:AtomList
rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil" />
          </expr:expressionBody>
      </expr:SWRL-Condition>
   </process:hasPrecondition>
</process:AtomicProcess>
<process:Input rdf:ID="Book">
    <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://127.0.
0.1/ontology/Concepts.owl#Book</process:parameterType>
    <rdfs:label></rdfs:label>
</process:Input>
<process:Output rdf:ID="PriceOfBook">
    <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://127.0.
0.1/ontology/Concepts.owl#Price</process:parameterType>
    <rdfs:label>Book Price</rdfs:label>
</process:Output>

<grounding:WsdlGrounding rdf:ID="BookPrice-Grounding">
    <service:supportedBy rdf:resource="#BookPrice-Service"/>
</grounding:WsdlGrounding>
</rdf:RDF>
```

Annexes D

Le listing suivant est un code java présentant une vue simplifiée de l'implémentation du moteur de recherche local. Nous avons utilisé la bibliothèque Jena pour automatiser l'inférence sémantique.

```
package mySearchEngine;
import com.hp.hpl.jena.ontology.OntClass;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.OntModelSpec;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.Property;
import com.hp.hpl.jena.rdf.model.RDFNode;
import com.hp.hpl.jena.rdf.model.Resource;
import com.hp.hpl.jena.rdf.model.StmtIterator;
import java.util.Vector;

public class mySearchEngine {

    static private final String W3C_UPPER_ONT_PROFILE =
"http://www.daml.org/services/owl-s/1.0/Profile.owl#";

    static private final String W3C_UPPER_ONT_PROCESS =
"http://www.daml.org/services/owl-s/1.0/Process.owl#";

    private OntModel m;     // représente un modèle d'ontologie dans Jena
    private String ontNS;   // namespace de l'ontology
    private String ontURL;  // URL du fichier qui contient l'ontologie
    // vont contenir les inputs et outputs de la requête
    Vector inputClass = new Vector();
    Vector outputClass = new Vector();
    Vector goalClass = new Vector ();

    /** Creer une instance de la classe mySearchEngine */
    public mySearchEngine(String ns, String url) {
        // notice that we don't need the user to specify the namespace:
        m = null;
        ontURL = url;
        ontNS = ns;
    // loadOntologyDocument();
    }
    // on va lire la requete du client
    // la requete est un fichier .owl donc il a une url

    public void readServiceRequest(String requestURL) {
        Model model = ModelFactory.createDefaultModel();
        model.read(requestURL);
        System.out.println(requestURL);
        String rightPrefix = "profile";
```

```
        if
(rightPrefix.equalsIgnoreCase(model.getNsURIPrefix(W3C_UPPER_ONT_PROFILE))
== false) {
            System.out.println(": is not a semnantic web service request
file!");
            return;
        } else {
            System.out.println(": is a semantic web service request
file.");
        }
        //trouver les inputs, les outputs et le but
        Vector inputName = new Vector();
        Vector outputName = new Vector();
        Vector goalName = new Vector();
        StmtIterator iter = model.listStatements();
        while (iter.hasNext()) {
        com.hp.hpl.jena.rdf.model.Statement stmt = iter.nextStatement();
        Resource subject = stmt.getSubject();
            Property predicate = stmt.getPredicate();
            RDFNode object = stmt.getObject();
            System.out.print("subject: " + subject.toString());
            System.out.print(", predicate: " + predicate.toString());
            System.out.println(", object: " + object.toString());
            String predicateString = predicate.toString();
            String objectString = object.toString();
            String subjectString = subject.toString();
            System.out.println("Object = " + objectString);
            System.out.println(objectString.equals(W3C_UPPER_ONT_PROCESS +
"Input"));
            if (objectString.equals(W3C_UPPER_ONT_PROCESS + "Input")) {
                String iName = null;
                iName =
subjectString.substring(subjectString.lastIndexOf("#") + 1);
                System.out.println("iName = " + iName);
                if (iName != null) {
                    if (inputName.contains(iName) == false) {
                        inputName.addElement(iName);
                    }
                }
            } else {
                if (objectString.equals(W3C_UPPER_ONT_PROCESS +
"UnConditionalOutput")) {
                    String oName = null;
                    oName =
subjectString.substring(subjectString.lastIndexOf('#') + 1);
                    if (oName != null) {
                        if (outputName.contains(oName) == false) {
                            outputName.addElement(oName);
                        }
                    }
                }
            }
        }
```

```
        //concepts en input
        String domainOntURL = null;
        for (int i = 0; i < inputName.size(); i++) {
            String currentInputName = inputName.elementAt(i).toString();
            iter = model.listStatements();

            while (iter.hasNext()) {

                com.hp.hpl.jena.rdf.model.Statement stmt =
iter.nextStatement();

                Resource subject = stmt.getSubject();

                Property predicate = stmt.getPredicate();

                RDFNode object = stmt.getObject();

                String predicateString = predicate.toString();

                String objectString = object.toString();

                String subjectString = subject.toString();

                System.err.println("object = " + objectString);

                System.err.println("current name = " + currentInputName);
System.err.println(subjectString.substring(subjectString.lastIndexOf("#") +
1));

if (subjectString.substring(subjectString.lastIndexOf("#") +
1).equals(currentInputName) &&

                        predicateString.equals(W3C_UPPER_ONT_PROCESS +
"parameterType")) {
                    if (domainOntURL == null) {
                        domainOntURL = objectString.substring(0,
objectString.lastIndexOf('#'));
                    }
inputClass.addElement(objectString.substring(objectString.lastIndexOf('#')
+ 1));
                    break;

                }

            }

        }

//Même chose pour les outputs et le but
 ……
        if (inputClass.size() == 0) {
            System.out.println("this service request does not require any
input parameters!");

        } else {
            System.out.print("          INPUT requested: ");
```

```
            for (int i = 0; i < inputClass.size(); i++) {
                System.out.print(inputClass.elementAt(i).toString() + ",
");
            }
        }
        System.out.println();
   //Même chose pour les outputs et le but
 ……
        }
        System.out.println();
        System.out.println();
        System.out.println("Le nombre de Inputs dans la requete = " +
inputClass.size());
        System.out.println("======================== finished reading the
service request ====================");
    }

    public Vector<myWebServiceDescription>
findQualifiedCandidates(Vector<myWebServiceDescription> allCandidates) {

        Vector<myWebServiceDescription> qualifiedCandidates = new
Vector<myWebServiceDescription>();

        System.out.println("\n" + "executing the matching algorithm...");

        m = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM_RULE_INF,
null);

        m.read(ontURL);  // add a protection here using addAltEntry()

        for (int k = 0; k < allCandidates.size(); k++) {
            // write out the current candidate
            // System.out.println(allCandidates.elementAt(i).toString());

            // get the service description from this candidate
            myWebServiceDescription sd = (myWebServiceDescription)
(allCandidates.elementAt(k));

            // if it is not the same domain-specific ontology, quit
            if (sd.getONTURL().equalsIgnoreCase(ontURL) == false) {
                System.out.println("l'ontologie generale est " + ontURL);
                System.out.println("l'ontologie du candidat est " +
sd.getONTURL());

                continue;   // continue to next candidate
            }

            // get input, output and goal from the current candidate
            Vector cInputs = sd.getInputs();
            Vector cOutputs = sd.getOutputs();
            Vector cgoal = sd.getgoal();
            if (cInputs.size() != inputClass.size()) {
                System.out.println(">>>>>> skip one candidate since the
number of input parameter is different!");
```

```
                System.out.println("Because the service has " +
cInputs.size() + " inputs");

                continue;  // continue to next candidate
            }

//Même chose pour les outputs et le but
 ……
…….
            // exact match possible?
            boolean exactMatch = true;
            for (int i = 0; i < inputClass.size(); i++) {
                //if ( cInputs.contains(inputClass.elementAt(i)) == false )
                String x = cInputs.get(i).toString();
                x = x.substring(x.lastIndexOf('#') + 1);
                System.out.println("****************************** le
service candiadat " + x);
                System.out.println("****************************** le
service requis  " + inputClass.elementAt(i).toString());
System.out.println(x.equals(inputClass.elementAt(i).toString()));

                if (!x.equals(inputClass.elementAt(i).toString())) {
                    exactMatch = false;
                    break;
                }
            }
//Même chose pour les outputs et le but
 ……
            if (exactMatch == true) {
                qualifiedCandidates.addElement(sd);
                continue;  // continue to next candidate
            }
……
    // décide si la première class (cn1) est une superclass de la deuxieme
classe (cn2)
    public boolean isAsuperB(String cn1, String cn2) {
        OntClass classA = m.getOntClass(ontNS + cn1);
        OntClass classB = m.getOntClass(ontNS + cn2);
        if (classA == null || classB == null) {
            return false;
        }
        return classB.hasSuperClass(classA, false);
    }
 // décide si la première class (cn1) est une sousclass de la deuxieme
classe (cn2)
    public boolean isAsubB(String cn1, String cn2) {
        OntClass classA = m.getOntClass(ontNS + cn1);
        OntClass classB = m.getOntClass(ontNS + cn2);
        if (classA == null || classB == null) {
            return false;
        }
        return classA.hasSuperClass(classB, false);
    }}
```

Annexes E

Le listing suivant présente le fichier BPEL correspondant au scénario schématisé dans la figure 5.5 du chapitre 5. Le nom du fichier est « FindPriceProcess.bpel »

```
<?xml version="1.0" encoding="UTF-8" ?>
- <process name="FindPriceProcess" ……..

  <import namespace="http://j2ee.netbeans.org/wsdl/FindPriceInDesiredCurrency"
     location="FindPriceInDesiredCurrency.wsdl"
     importType="http://schemas.xmlsoap.org/wsdl/" />

  <import …… location="localhost_8080/FindBookPriceServicerService/
     FindBookPriceServicerWrapper.wsdl"
     importType="http://schemas.xmlsoap.org/wsdl/" />

<import ……

<import ……

<import ……

<! Importer les contrats WSDL des 4 Web services (WS1= FindPriceLocalPL, WS2=
     FindBookPriceServicer, WS3=ReserveConvertor and WS4= OtherPeerConvertorPL)

- <partnerLinks>

  <partnerLink name=" FindPriceReservePL"
     partnerLinkType="tns:FindBookPriceServicerLinkType"
     partnerRole="FindBookPriceServicerRole" />

  <partnerLink name="FindPriceLocalPL"
     partnerLinkType="tns:FindPriceLocalServiceLinkType"
     partnerRole="FindPriceLocalServiceRole" />

  <partnerLink name="OtherPeerConvertorPL"
     partnerLinkType="tns:ConvertorServiceLinkType"
     partnerRole="ConvertorServiceRole" />

  <partnerLink name="ReserveConvertor" ReserveConvertor
     partnerLinkType="tns:ConvertisseurLinkType" partnerRole="ConvertisseurRole" />

  <partnerLink name="PartnerLink1"
     partnerLinkType="tns:FindPriceInDesiredCurrency"
     myRole="FindPriceInDesiredCurrencyPortTypeRole" />

    </partnerLinks>

- <variables>

  <variable name="FindPriceInDesiredCurrencyOperationOut6"
     messageType="tns:FindPriceInDesiredCurrencyOperationResponse" />
```

```
<variable name="ConvertirLocalOut2" xmlns:tns="http://service/"
  messageType="tns:convertirResponse" />

<variable name="ConvertirLocalIn2" xmlns:tns="http://service/"
  messageType="tns:convertir" />

<variable name="FindPriceInDesiredCurrencyOperationOut5"
  xmlns:tns="http://j2ee.netbeans.org/wsdl/FindPriceInDesiredCurrency"
  messageType="tns:FindPriceInDesiredCurrencyOperationResponse" />

<variable name="ConvertOtherPeerOut2" xmlns:tns="http://net/"
  messageType="tns:convertResponse" />

<variable name="ConvertOtherPeerIn2" xmlns:tns="http://net/"
  messageType="tns:convert" />

……<! Déclaration des variables

  ……

  </variables>

- <sequence>

<receive name="Receive1" createInstance="yes" partnerLink="PartnerLink1"
  operation="FindPriceInDesiredCurrencyOperation"
  xmlns:tns="http://j2ee.netbeans.org/wsdl/FindPriceInDesiredCurrency"
  portType="tns:FindPriceInDesiredCurrencyPortType"
  variable="FindPriceInDesiredCurrencyOperationIn" />

- <assign name="Assign1">

- <copy>

<from variable="FindPriceInDesiredCurrencyOperationIn" part="ISBN" />

<to>$FindBookPriceOtherPeerIn.parameters/isbn</to>

  </copy>

  </assign>

<invoke name="InvokeFindPriceReserve" partnerLink="FindPriceReservePL"
  operation="findBookPrice" xmlns:tns="http://services/"
  portType="tns:FindBookPriceServicer" inputVariable="FindBookPriceOtherPeerIn"
  outputVariable="FindBookPriceOtherPeerOut" />

- <if name="If1">

<condition……

……// déroulement du processus selon la priorité d'exécution des
  Web services (exécutés et en réserves)
```

.....

```
<reply name="Reply6" partnerLink="PartnerLink1"
    operation="FindPriceInDesiredCurrencyOperation"
    xmlns:tns="http://j2ee.netbeans.org/wsdl/FindPriceInDesiredCurrency"
    portType="tns:FindPriceInDesiredCurrencyPortType"
    variable="FindPriceInDesiredCurrencyOperationOut6" />

    </sequence>

    </else>

    </if>

    </sequence>

    </else>

    </if>

    </sequence>

    </else>

    </if>

    </sequence>

    </process>
```

Annexes F

Cette annexe contient une petite présentation de la plateforme JXTA. Plus de détails sont disponible dans **[GRA02]**.

1. Définition

JXTA (à prononcer juxta) est une plateforme de Sun dédiée au développement des applications P2P. Elle comporte une suite de protocoles Open Source.

2. Terminologies

Les concepts de base de JXTA sont les suivants :

Identifiant : Les différentes entités ont un identifiant unique, codé sur 128 bits *UUID*. JXTA introduit un ID appelé URN. Il s'agit d'une chaîne de caractère unique, utilisée pour identifier les différents types de ressources tels que: les pairs, les groupes de pairs, les tubes de communications, ….

Pair : Un pair est une entité qui peut communiquer à travers les protocoles requis. Il peut s'agir de n'importe quel type de composant réseau, depuis le PDA jusqu'au super ordinateur.

Groupes de pairs : Un groupe de pairs est une entité virtuelle qui peut communiquer à travers l'ensemble des protocoles Un groupe de pairs est une collection de pairs offrant le même type de services, chacun étant identifié par un ID unique.

Pipes : JXTA utilise les pipes comme mécanisme de communication. Ils sont des tubes de communication pour envoyer et recevoir des messages. On repère des pipes d'entrée et des pipes de sortie.

Annonce : Une annonce (avertissement) est un document XML qui nomme, décrit et publie l'existence d'une ressource. Toutes les ressources JXTA, telles que les pairs, les groupes et les tubes de communications, sont représentés par des annonces.

Message : Un message est un objet qui est transmis entre des pairs JXTA. Il est composé d'une enveloppe et d'un corps.

La structure de JXTA est constituée de 3 couches :

Le noyau : JXTA Core

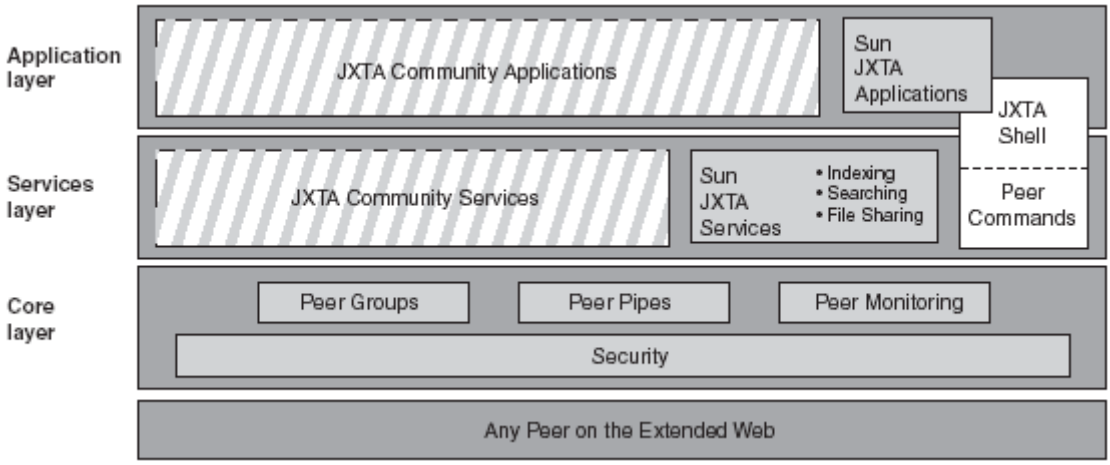
Le noyau implémente les fonctionnalités de base permettant d'avoir un réseau Peer-to-Peer (normes, standards,…). Il permet la communication, le monitoring ou encore la sécurité.

La couche des services : JXTA Services

Cette couche propose différents services qui utilisent les protocoles de la couche inférieure pour accomplir une tâche, comme par exemple la recherche, le partage de fichiers, l'authentification…

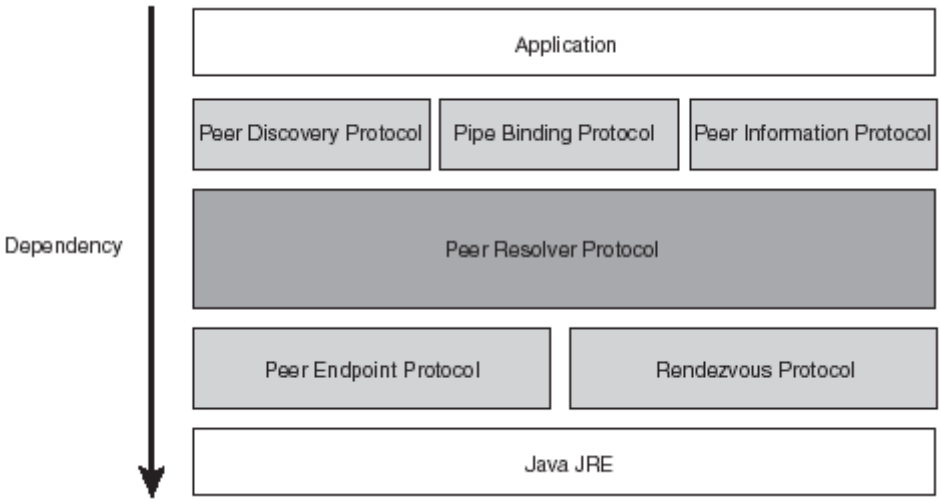
La couche des applications : JXTA Applications

La couche des applications est la couche la plus haute, où viennent se placer les différentes applications utilisant des services de la couche précédente.



3. Les protocoles de JXTA

JXTA propose une suite de protocoles développés indépendamment de tout langage. Cette suite est actuellement composée de six protocoles :



- **Peer Resolver Protocol (PRP)** : il est utilisé pour envoyer une requête à un nombre indéterminé de pairs et recevoir une réponse.
- **Peer Discovery Protocol (PDP)** : ce protocole est utilisé pour annoncer et découvrir du contenu. Il peut être utilisé pour chercher n'importe quel pair, groupe de pairs ou annonce.

- **Peer Information Protocol (PIP)** : utilisé pour obtenir l'état d'un autre pair. Par exemple l'envoi d'un message ping pour savoir si un pair existe encore.
- **Pipe Binding Protocol (PBP)** : utilisé pour créer un chemin de communication, en faisant correspondance entre les entrées et les sorties des pipes.
- **Peer Endpoint Protocol (PEP)** : utilise pour trouver une route entre deux pairs.
- **Rendezvous Protocol (RVP)** : utilise pour propager les messages dans le réseau.

Résumé *— Depuis la naissance de l'Architecture Orientée Services (SOA), la découverte a présenté un problème important pour la composition automatique des Web services. Ces derniers doivent interagir dynamiquement avec le minimum d'intervention humaine. Ils devraient être capables de découvrir d'autres services qui ont des capacités bien définies et qui réalisent des tâches précises. Pour améliorer la découverte automatique des Web services, différentes technologies sont utilisées pour fournir des nouvelles solutions. Dans ce contexte, le Web sémantique et les systèmes P2P sont les technologies les plus adaptées aux approches orientées services. Dans ce travail, nous présentons une stratégie basée sur un algorithme épidémique pour la découverte et la composition des Web services sémantiques dans les réseaux P2P non structurés. Dans cette stratégie, nous utilisons une table distribuée pour préserver les traces des compositions déjà réalisées dans le réseau. De plus, nous présentons une architecture distribuée pour implémenter cette solution. Une étude de cas est présentée pour valider l'approche proposée.*

Mots clés : *Web services, Web sémantique, P2P, découverte et composition des Web services*

Abstract *— Since the beginning of the Service Oriented Architecture (SOA), the Web service discovery presents an important problem for Web services composition. Web Services should act autonomously with as minimal human intervention as possible, they should be able to discover other services which have particular capabilities and realize precise tasks. To improve the automatic discovery of Web services, different technologies are used to provide new flexible solutions for web services discovery and composition. In this context, the Web semantic and the Peer-to-Peer (P2P) computing are the most adapted technologies to the service-oriented approaches. In this work, we present a distributed strategy based on epidemic algorithms to discover semantic Web services in unstructured P2P networks. In this strategy, we use a distributed table to preserve the composition way of the P2P composed Web services. Also, we present a distributed architecture for which implements this solution. A case study is presented to improve the proposed model.*

Keywords: *Web services, Semantic Web, P2P, Web services discovery and composition*

ملخص- *منذ ظهور الهندسة المعتمدة على الخدمات، شكل اكتشاف خدمات الواب مشكلا مهما لعملية تركيب خدمات الواب. هذه الأخيرة يجب أن تتفاعل تلقائيا بدون تدخل بشري قدر الإمكان. يجب عليها أن تكتشف خدمات أخرى لديها قدرات معينة و تقوم بمهام محددة. من أجل تحسين عملية الاكتشاف التلقائي لخدمات الواب، هناك تكنولوجيات أخرى تم استعمالها من أجل إحكام عمليات اكتشاف و تركيب خدمات الواب. في هذا الخضم، الواب المعنوي و أنظمة الند للند تعتبر التكنولوجيات الأكثر ملائمة للحلول المعتمدة على خدمات الواب. في هذا العمل قدمنا إستراتجية تعتمد على خوارزم وبائي من أجل اكتشاف خدمات الواب المعنوية في شبكات الند للند غير المهيكلة. في هذه الإستراتجية استعملنا جدول موزع من أجل حفظ طرق تركيبات الخدمات السابقة التي تم إتمامها في شبكة الند للند. بالإضافة إلى ذلك قدمنا هندسة موزعة من أجل تحقيق الحل المقدم. في الأخير تم تقديم دراسة خاصة من أجل إثبات الحل المعروض.*

كلمات مفتاح: *خدمات الواب، الواب المعنوي، الند للند، اكتشاف و تركيب خدمات الواب*