Contributions à la description et la découverte de services web sémantiques

Yassin Chabeb

TELECOM SudParis

Université d'Évry Val d'Essonne

Thèse de doctorat de Télécom SudParis dans le cadre de l'école doctorale S&I en co-accréditation avec l' Universite d'Evry-Val d'Essonne

Spécialité :
Informatique

Par :

Mr. Yassin CHABEB

Thèse présentée pour l'obtention du diplôme de Docteur
de Télécom SudParis

Contributions à la Description et la Découverte de Services Web Sémantiques

Soutenue le 23 novembre 2011 devant le jury composé de :

Rapporteurs :

Mme.	Marie-Christine FAUVET	Professeur à l'Université Joseph Fourier de Grenoble - LIG
M.	Thierry DELOT	Maître de Conférences (HDR) à l'Université de Valenciennes - LAMIH

Examinateurs :

Mme.	Hanna KLAUDEL	Professeur à l'Université d'Evry-Val d'Essonne - IBISC (Présidente du jury)
M.	Hugues VINCENT	Ingénieur de recherche à Thales Communications France
M.	Bruno DEFUDE	Professeur à TELECOM SudParis - SAMOVAR
M.	Samir TATA	Professeur à TELECOM SudParis - SAMOVAR (Directeur de thèse)

Thèse numéro : *2011TELE0026*

Remerciements

Le travail que j'ai réalisé au cours de ma thèse n'aurait certainement pas été possible sans l'aide, les encouragements et les conseils de nombreuses personnes qui y ont été impliquées de près ou de loin. C'est pourquoi je tiens à leur exprimer par ces quelques lignes toute ma reconnaissance, en particulier :

Je suis très reconnaissant envers Mme. Marie-Christine FAUVET, Professeur à l'Université Joseph Fourier de Grenoble et M. Thierry DELOT, Maître de Conférences (HDR) à l'Université de Valenciennes pour avoir accepté de rapporter mes travaux. Je les remercie d'avoir apporté tant de soins à la relecture de ce mémoire et d'avoir accepté de figurer dans mon jury.

Je tiens à exprimer ma gratitude à Mme. Hanna KLAUDEL, Professeur à l'Université d'Évry-Val d'Essonne et à M. Hugues Vincent, Ingénieur de recherche à Thales Communications France pour avoir accepté d'examiner mes travaux et faire partie du jury.

Je remercie mon premier directeur de thèse Bruno Defude pour m'avoir accueilli dans le département Informatique et m'avoir permis d'effectuer cette thèse dans de bonnes conditions. J'ai particulièrement apprécié sa disponibilité, son ouverture d'esprit et les nombreuses discussions qui ont animé ces années de thèse.

Je remercie chaleureusement mon directeur de thèse Samir Tata pour m'avoir encadré tout au long de cette thèse. Je le remercie pour ses conseils, sa disponibilité, sa patience et son dynamisme qui m'ont été très profitables et qui m'ont permis d'avancer tout au long de cette thèse. J'ai beaucoup appris en travaillant avec lui.

Je tiens également à remercier les membres du département Informatique et de la direction de Recherche pour leur aide et collaboration, en particulier Mme. Xayplathi Lyfoung, M. Djamel Belaid et notre chère assistante de gestion Brigitte Houassine. Je remercie Mohamed Sellami, Zied Abid et Léon Lim pour avoir patiemment lu et corrigé certaines parties de ce document. Je remercie Walid Gaaloul, Amel Bouzgoub et Denis Conan pour les conseils et les encouragements. Merci également à mes collègues et amis Mohamed Abid, Walid Achour, Tarek Abidi, Gabriel Adgeg, Rami Amri, Sandrine Beauche, Imen Benzarti, Djallel Bouneffouf, Imen Brahem, Marie Buffat, Firas Fki, Mohamed-Taher Hassani, Anis Jdidi, Imen Lahmer, Mohamed Loussaief, Mohammed El-Amine Matougui, Youssouf Mhoma, Chan Nguyen, Sadok Mazigh, Nomane Ould-Ahmed Mbarek, Alain Ozanne, Amin Sakka, Rami Sellami, Mouna Selmi, Jerome Sicard, Sofien Somai, Dorsaf Zekri et à tous les autres pour leur aide amicale et permanente, et pour les moments de rire et de détente.

Merci aux membres de l'Unité de Recherche UTIC de l'ESSTT à l'origine de ce long parcours académique, en particulier Professeur Mohamed Jemni, Oussama El-Ghoul et tous mes anciens enseignants et collègues. Merci pour Professeur Fathi Debili, Professeur Christian Fluhr et Zied Ben Tahar pour leurs précieuses aides. Merci à mes amis pour leur soutien, leur présence, leur compréhension et dont l'amitié m'a apporté les moments de réconfort et de distraction nécessaires lors du déroulement d'un tel projet. J'adresse un remerciement particulier Dorra Abidi, Thierry Barisien, Dr. Salah Zitouni et Tayeb Zitouni qui m'ont soutenu durant ces années. Merci à ma patrie, la Tunisie, de m'avoir offert l'occasion de continuer mes études à Paris, et je souhaite la réussite de sa révolution.

Un grand merci à ma famille qui m'a apportée le soutien, la confiance et l'amour indéfectible que j'espère rendre pareillement et transmettre à mon tour. Merci à mes parents, feu mon père Hamadi Chabeb et ma mère Fatma Zitouni, de m'avoir offert l'occasion de faire des études longues, qui ont abouti à cette thèse.

Et par dessus tout, je réserve une pensée particulière à ma chère femme Layla pour avoir toujours su trouver les mots qu'il faut dans les meilleurs comme dans les pires moments et pour m'avoir soutenu, encouragé et supporté durant ces années de thèse. Dieu Merci de m'avoir accorder ton aide à travers toutes ces personnes.

Résumé

Les travaux de recherche menés autour de la description de services Web utilisent de plus en plus des modèles sémantiques pour fournir une représentation interprétable automatiquement. Toutefois, nous avons décelé des lacunes dans les approches sémantiques actuelles qui engendrent ambiguïté et non pertinence au niveau de l'appariement et de la découverte de services Web.

Pour remédier à ces lacunes nous proposons des contributions à la description et à la découverte de services Web sémantiques. En ce qui concerne la description de services, nous avons défini un langage basé sur une recommandation W3C. En plus d'une annotation métier sémantique des éléments d'un service, notre principale contribution à la description sémantique consiste à spécifier la nature de ces annotations en utilisant une ontologie technique que nous avons définie. Cette ontologie met en relation plusieurs concepts sémantiques de services Web que nous avons identifiés dans des approches existantes et intègrera d'autres concepts qu'on définira ultérieurement sans pour autant modifier notre langage de description ou nos techniques d'appariement associées.

Nous avons également défini un algorithme d'appariement entre une requête de service et les descriptions des services publiés. Cet algorithme se base sur un appariement entre éléments d'une requête et un service publié et trois techniques d'agrégation des résultats d'appariements élémentaires. L'algorithme tire avantage de la description sémantique que nous avons définie. Il a été mis en œuvre dans un annuaire de services Web sémantiques et a été également comparé aux algorithmes de référence. Les expérimentations montrent clairement l'efficacité de notre approche en termes de temps de réponse et de précision.

Mots-clés : Services Web Sémantiques, Description de Services, Découverte de Services, Matching de Services, Ontologie.

Abstract

Researchs conducted around Web service description use more and more of semantic models to provide an automatically interpretable representation. However, we identified gaps in current approaches that generate semantic ambiguity and impertinence at Web service matching and discovery.

To address these shortcomings we propose contributions about semantic Web service description and discovery. As for the Web services description, we have defined a language based on a W3C Recommendation. In addition to a semantic business annotation of service components, our main contribution about the semantic description is to specify the nature of these annotations using a technical ontology that we have defined. This ontology merges several semantic concepts of web services that we identified in existing approaches and may include other concepts that can be defined later without changing our description language or our matching techniques.

We also defined a matching algorithm between a service request and published service descriptions. This algorithm is based on matching between elements of a service request and descriptions of published services. This matching is may be computed by three aggregation techniques of the results of those elements' matching. The algorithm takes advantage of the semantic description we have defined. It was implemented in a semantic web services registry and was also compared to referenced algorithms. The experiments clearly demonstrate the effectiveness of our approach in terms of response time and precision.

Keywords : Semantic Web Services, Web Service Description, Web Service Discovery, Semantic Matching, Ontology.

Table des matières

Table des figures

Liste des tableaux

Première partie

Introduction générale

Introduction

Contexte et problématique

De nos jours, le domaine des services Web et le domaine du Web sémantique se croisent de plus en plus. Les services Web ont permis au Web de passer du rôle habituel de réseaux d'informations à un intergiciel (*Middleware*) d'applications grâce à l'exploitation croissante des technologies XML. En effet, les services Web étant des applications modulaires, faiblement couplées et auto-descriptives, ils parviennent à fournir un modèle facile à comprendre et à utiliser dans la programmation et le déploiement d'applications. En plus, le modèle est basé sur des normes et est apte à s'exécuter à travers le Web [35].

Ce nouveau rôle d'intergiciel d'applications fortement intéressant pour le secteur B2B (*Business-To-Business*) est avantagé par l'utilisation de protocoles et de langages standards. Il est vrai que ces derniers permettent plus d'interopérabilité entre applications provenant de différentes plates-formes mais certaines lacunes ont vite fait surface. Souvent, durant la découverte, des exigences sémantiques sont relevées. Cependant, ni les normes (telles que *SOAP* [3] et *UDDI* [3]), ni les langages de description (tel que *WSDL* [15]) ou de modélisation de processus métiers (tel que *WS-BPEL* [89]) ne considèrent l'enjeu de la sémantique. En fait, *SOAP* n'est qu'un protocole de transport de messages XML entre les services Web. *UDDI* est un modèle de description homogène d'annuaires de services Web. Basé sur XML, *WSDL* est un standard de description qui reste purement syntaxique. Finalement, *WS-BPEL* est le standard de modélisation de processus métiers et d'orchestration de services Web qui se focalise sur la syntaxe.

Le manque de sémantique dans les langages de description et de modélisation pénalise le catalogage automatique des descriptions de services pendant la phase de publication ainsi que l'appariement automatique entre descriptions de services pendant la phase de découverte. En effet, d'éventuels problèmes d'hétérogénéités sémantiques se posent lors de la découverte, l'invocation voire la conception des processus métier. Les problèmes de découverte surviennent au moment où on soumet une demande présentée sous forme de requête sur des services publiés dans un annuaire et qu'on n'identifie pas ou on identifie mal le degré de similarité entre les descriptions de service requis (la requête) et celles des services offerts (les services publiés). Par exemple pour *WSDL*, tous les principaux éléments des services doivent être pris en considération dans cette procédure, à savoir : *interface, operation, input* et *output*. Des problématiques telles que l'imprécision dans la découverte et l'incomplétude de la description peuvent apparaître dans le mécanisme d'évaluation de la similarité pendant la découverte, qui est appelé *Matching* (Appariement).

Le Web sémantique peut fournir des solutions à de telles exigences d'interopérabilité et de précisions sémantiques. Les machines peuvent y coopérer grâce à un contenu de pages Web

décrit explicitement avec une sémantique basée sur des ontologies de référence. En effet, c'est ce qui permet aujourd'hui que les données soient compréhensibles autant par des agents logiciels que par des humains [5].

L'application des principes du Web sémantique aux services Web a permis de profiter de la considération du sens et de la signification des données échangées pour améliorer la faisabilité des différentes tâches : description, appariement, découverte, etc. En effet, cet aspect offrait aux services Web, d'une part, une description sémantique explicite de leurs fonctionnalités aussi compréhensible par les agents logiciels que par les utilisateurs humains, et d'autre part, une interprétation correcte des informations envoyées et reçues.

Les travaux de recherche menés autour de la description des services Web utilisent de plus en plus les ontologies pour fournir une représentation de l'information sémantique, à la fois, détaillée, riche et facile à manipuler par les machines. Les ontologies permettent d'améliorer la description et la découverte de services Web. Les approches sémantiques actuelles présentent différentes lacunes qui engendrent des ambiguïtés et des imprécisions pour l'appariement des éléments des descriptions lors de la découverte. Une fois ces lacunes comblées, nous pensons qu'on surmontera plusieurs limitations des algorithmes de découverte et des architectures d'annuaires sémantiques.

Nos travaux de recherche s'intéressent aux architectures orientées services sémantiques, dans lesquelles les fournisseurs de services Web peuvent décrire, publier et découvrir des Services Web Sémantiques. Ces services sont accessibles et consultables en ligne depuis un annuaire sémantique. Notre approche vise à la proposition de techniques sémantiques pour rendre les processus de description, stockage et découverte de services efficaces en termes de temps d'exécution et précis en ce qui concerne la qualité de découverte. Pour cela, nous examinons les différents aspects de la problématique de description sémantique et de découverte sémantique en termes qualitatif et quantitatif. Certaines approches de description sémantique présentent quelques lacunes en termes d'expressivité et d'extensibilité, c'est-à-dire, sur leur aspect qualitatif. Les approches de découverte sont d'avantage jugées sur des questions de précision, de rappel et de temps de réponse.

Objectifs

Notre premier objectif est de proposer une approche pour la description sémantique de services Web. Compte tenu du contexte de notre travail et des besoins décrits ci-dessus, nous avons identifié deux points à considérer comme sous-objectifs :

- Proposer une approche de description qui adapte le minimum possible les standards et réutilise les outils déjà disponibles. Nous visons une approche issue des standards et des recommandations (e.g. WSDL, SAWSDL, OWL-S) afin d'enrichir la spécification d'informations sémantiques qui décrivent les services Web.
- Proposer une description autant explicite que précise en termes de sémantique, qui couvre les principaux éléments fonctionnels et qui décrit la fonctionnalité avec le plus possibles d'éléments tout en ayant la capacité d'ajouter des éléments non-fonctionnels et fonctionnels ultérieurement.

Les deux sous-objectifs visent une contribution guidée plutôt par l'aspect qualitatif (réutilisation, extensibilité, expressivité).

Notre deuxième objectif consiste à proposer un algorithme d'appariement pertinent entre une requête de service et les descriptions de services publiés. Compte tenu du contexte de notre travail et des besoins décrits ci dessus, nous avons identifié deux points à considérer comme sous-objectifs :

- Proposer une approche d'appariement pertinente. La précision des techniques d'appariement est primordiale pour évaluer un certain aspect de la pertinence d'une approche : plus on réussit à apparier d'éléments des deux services, requis et offert, plus le résultat de découverte est pertinent, et plus on identifie avec précision la similarité entre les éléments des deux services, requis et offert, plus le résultat de découverte est pertinent.
- Proposer une approche d'agrégation efficace. L'agrégation des résultats d'appariements élémentaires (ou plus généralement partiaux) doit être bien définie afin de calculer de manière pertinente le degré final/global d'appariement pour la découverte.

Les deux sous-objectifs visent une contribution guidée plutôt par l'aspect quantitatif (précision, rappel et de temps de réponse).

Notre troisième objectif porte sur une proposition de stockage sémantique de services Web qui supporte nos propositions de description sémantique de services Web et de découverte sémantique de services Web.

Approche

Afin d'atteindre notre premier objectif, nous proposons une approche de description sémantique de services Web qui repose sur les points suivants :

- Une description de services qui ne dépend pas d'un langage d'ontologie en particulier pour décrire l'aspect sémantique du service Web,
- L'adoption des standards et la réutilisation des outils de services Web déjà disponibles,
- Une description de service qui décrit la fonctionnalité avec le plus possibles d'éléments tout en ayant la capacité d'ajouter des éléments non-fonctionnels et fonctionnels ultérieurement,
- Une description sémantique autant explicite que précise et qui couvre les éléments fonctionnels les plus importants.

Afin d'atteindre notre deuxième objectif, nous proposons une approche de découverte et d'appariement sémantique qui reposent sur les points suivants :

- Un algorithme d'appariement avec des techniques d'appariement qui parcourt la plupart des éléments des deux services, requis et offert,
- Un mécanisme d'appariement qui identifie précisément et fidèlement la similarité entre les éléments des deux services, requis et offert,
- Un algorithme d'agrégation efficace qui calcule de manière pertinente le degré final/global d'appariement pour la découverte (via des solutions arithmétiques ou basées sur les graphes des descriptions de services Web).

Afin d'atteindre notre troisième objectif, nous proposons un annuaire sémantique de services Web qui repose sur les points suivants :

- Un annuaire qui renforce la prise en charge de la sémantique avec des composants d'architecture bien dédiés à chaque aspect de stockage, d'appariement et de découverte,

– Un annuaire sémantique qui se base sur l'automatisation des phases de publication et de découverte pour améliorer les performances et réduire au maximum l'intervention d'utilisateur dans la découverte.

Plan du document

Ce document est composé d'une introduction générale, de sept chapitres et d'une conclusion générale.

Le chapitre 1 présente un état de l'art de la description de services. Dans ce chapitre, nous commençons par présenter le standard des services Web WSDL : les principes et les détails de ses deux spécifications les plus utilisées. Ensuite, nous présentons les approches SAWSDL, OWL-S et WSMO, nous finissons par l'approche USDL.

Le chapitre 2 présente un état de l'art des approches d'annuaires et de plate-formes de stockage de descriptions de services. Ce chapitre présente les différentes approches et modèles ainsi que les plate-formes les plus connues de stockage de descriptions de services que nous considèrons dans nos travaux.

Le chapitre 3 constitue un état de l'art de la découverte de services. Nous présentons dans ce chapitre les différentes approches et modèles ainsi que les plate-formes les plus connues de découverte de services. Nous commençons par exposer les approches de découverte non-logiques (section 3.2), puis les approches de découverte logiques (section 3.3) et finalement les approches de découverte hybrides (section 3.4). Une synthèse est présentée à la fin du chapitre (section 3.5), dans laquelle j'identifie notamment les limitations de certaines approches vis-à-vis du problème de la découverte de services sémantiques.

Le chapitre 4 est consacré à notre contribution à la description de services. Nous présentons tout d'abord les motivations de cette contribution. Nous présentons les principes du langage de description sémantique YASA, les avantages et l'utilisation de l'approche. Ensuite, nous présentons l'Ontologie Technique des Services Web sémantiques OTES, utilisée dans l'approche YASA.

Le chapitre 5 présente notre contribution à l'appariement et la découverte de services Web sémantiques. Dans ce chapitre, nous présentons les bases d'appariement sémantique élémentaire, les principes de nos approches d'agrégation sémantique pour l'appariement global. Cette approche nous a permis de concevoir et réaliser un nouveau matchmaker (apparieur) sémantique pour les services Web. Ce matchmaker sémantique est adapté aux services Web décrits en standard *WSDL*, en *SAWSDL* et en *YASAWSDL* [11].

Le chapitre 6 présente l'annuaire sémantique de services Web. Dans ce chapitre, nous commençons par présenter l'architecture de l'annuaire des services Web sémantiques. Ensuite, nous décrivons comment notre annuaire sémantique assure la phase de publication et de stockage des descriptions sémantiques de services. Finalement, nous nous focalisons sur la phase d'appariement et de découverte.

Le chapitre 7 présente nos réalisations à travers la préparation d'un banc d'essai, des évaluations de performances de la découverte, une vue globale du travail de développement et déploiement ainsi qu'une exploitation de nos contributions dans une étude de cas.

Deuxième partie

État de l’art

Chapitre 1

Description de Services Web Sémantiques

Sommaire

1.1 Introduction

Le domaine de services Web trouve de plus en plus de l'intérêt dans le domaine du Web sémantique. De même, grâce aux services Web, le Web a acquis la fonctionnalité d'intergiciel (*Middleware*) d'applications. Avant cela, il ne jouait qu'un rôle habituel de réseaux d'informations, ceci est surtout assuré par l'explosion de l'utilisation des technologies XML. Les services Web comme étant des applications modulaires, faiblement couplées et auto-descriptives parviennent à fournir un modèle facile à comprendre et à utiliser dans la programmation et le déploiement d'applications. Le modèle est apte à s'exécuter au travers le Web tout en se basant sur des normes [35].

Ce nouveau rôle d'intergiciel d'applications utile au secteur B2B (*Business-To-Business*) a comme avantage l'utilisation de protocoles et de langages standards. Il est vrai que ces derniers

permettent plus d'interopérabilité entre applications provenant de différentes plate-formes mais certaines lacunes ont vite fait surface. Citons quelques exemples, la manque de la sémantique dans le standard de description de services Web et l'ambiguïté dans la description de services Web spécifiée selon certaines approches sémantiques. Souvent, durant la découverte ou la composition, des exigences sémantiques sont relevées, telles que l'enrichissement des mécanismes d'annotation sémantique et ce qu'il engendrera d'amélioration des mécanismes de découverte. Cependant, ni les normes (telles que *SOAP* [3] et *UDDI* [3]), ni les langages de description ou de modélisation de processus métiers (tels que *WSDL* [15] et *WS-BPEL* [89]) ne considèrent l'enjeu de la sémantique. En fait, *SOAP* n'est qu'un protocole de transport de messages XML entre les services Web. *UDDI* est un modèle de description homogène d'annuaires de services Web. Basé sur XML, *WSDL* est un standard de description qui reste purement syntaxique. Finalement, *WS-BPEL* est le standard de modélisation de processus métiers et l'orchestration de services Web qui focalise sur la syntaxe. Ce manque de sémantique dans les langages de description et de modélisation pénalise à la fois la découverte et l'orchestration automatiques. En effet, d'éventuels problèmes d'hétérogénéités sémantiques s'imposent aux moments de découverte, invocation voire conception des processus métier. La réponse à de telles exigences d'interopérabilité sémantiques a été retrouvée dans le Web sémantique. Les machines peuvent y coopérer grâce à un contenu de pages Web décrit explicitement avec une sémantique basée sur des ontologies de référence. En effet, ceci rend aujourd'hui les données compréhensibles autant par des agents logiciels que par des humains [5].

L'application des principes du Web sémantique au domaine des services Web permet d'utiliser le sens et la signification des données échangées pour améliorer la faisabilité des différentes tâches : découverte, composition, etc. En effet, cet aspect offre aux services Web, d'une part, une description sémantique explicite de leurs fonctionnalités aussi compréhensible par les agents logiciels que par les utilisateurs humains, et d'autre part, une interprétation correcte des informations envoyées et reçues dans le cadre de découverte, d'invocation et surtout de composition et d'orchestration.

De nombreux langages et approches ont été développés dans l'objectif de décrire les services Web sémantiques. Dans ce chapitre, je passe en revue les plus connues parmi eux à savoir *SAWSDL, OWL-S, WSMO* et *USDL*. Le reste de ce chapitre est organisé comme suit. Dans la section 1.2.1, je commence par présenter le standard des services Web WSDL : les principes et les détails de ses deux spécifications les plus utilisées. Ensuite, je présente les approches SAWSDL (section 1.2.2), OWL-S (section 1.3.1) et WSMO (section 1.3.2). Et je finis par l'approche USDL (section 1.2.3). Enfin, je propose une synthèse et une conclusion pour le chapitre.

1.2 Approche de description à base d'annotations

1.2.1 Le standard WSDL

Le besoin d'une description claire de la communication entre services Web a abouti à des normalisations aux niveaux des messages échangés et des protocoles. Une grammaire XML bien structurée était proposée pour décrire les services Web et paramétrer les échanges de messages. Un langage de description s'est basé sur cette grammaire et est devenu le standard des services Web, c'est le "*Web Service Description Language*" (WSDL) [14]. Dans la suite, je présente les principes du standard WSDL et ces deux spécifications largement utilisées par les chercheurs et les industriels.

Principes du standard

Sur le Web ou dans un réseau, des applications peuvent communiquer grâce aux services. Ces communications ont pu être automatisées grâce au standard WSDL et son approche de description contenant les données nécessaires aux échanges entre les applications. Pour ce faire, WSDL fournit, à la fois, un format XML et un modèle pour décrire les services Web. Il permet de séparer la description de la fonctionnalité abstraite offerte par un service de ses détails concrets, tels que comment et où peut-on accéder à la fonctionnalité [13]. Le standard offre une description à deux parties : une première partie de description abstraite et une deuxième partie de description concrète.

Les éléments de description représentent des concepts du modèle WSDL. Les éléments de la partie abstraite décrivent principalement le service en termes de messages (envoyés et reçus). Ils sont indépendamment décrits par un schéma XML (au niveau d'une balise XML notée "*types*"). La partie abstraite inclut des éléments notés "*operation*" dont chacune associe un ou plusieurs messages à un modèle d'échange de messages. Le modèle spécifie le nombre et l'ordre ainsi que la cible et la source abstraites des messages. Les opérations sont groupées dans un élément noté "*interface*" ou aussi *portType*" sans préciser le moyen (le protocole) d'échange de messages.

En effet, afin qu'un client puisse communiquer avec un service Web décrit par une telle interface abstraite, il doit savoir comment et vers où les messages sont envoyés sur le réseau. C'est la partie concrète qui précise les détails sur le moyen d'échange de messages d'une ou plusieurs interface. Tout est spécifié par les éléments "*binding*".

Techniquement, la structure d'un élément "*binding*" ressemble à la structure d'un élément "*interface/portType*" mais avec des détails en plus. La spécification WSDL offre deux types de "*binding*" pour l'envoi de messages entre client et services Web : soit intégrés dans des messages du protocole SOAP transmis par exemple en HTTP, soit directement par exemple dans des messages du protocole HTTP.

Une adresse réseau peut être associée à un "*binding*" grâce au concept de "*endpoint/port*". En effet, un élément XML noté "*service*" groupe ensemble les "*endpoint*"s implémentant une même interface [13]. Ainsi, un service Web peut être accessible depuis plusieurs "*endpoint*"s. Chacun est potentiellement associé à un "*binding*" différent. Je prend l'exemple typique d'un premier "*endpoint*" qui utilise un protocole de transmission sans chiffrement (pour un réseau intranet sûr) et d'un deuxième "*endpoint*" qui utilise SOAP sur HTTP (pour un accès Internet) [94].

Genèse des éléments de description

Développé par IBM, Microsoft et Ariba en septembre 2000, WSDL 1.0 combinait les deux langages de description de services : NASSL (*Network Application Service Specification Language*) de IBM et SDL (*Service Description Language*) de Microsoft. Bien que la formalisation WSDL 1.1, publiée en mars 2001, n'a pas été approuvée par le W3C, elle est devenue largement utilisée. En juin 2003, la version WSDL 1.2 était juste une ébauche W3C. Bien que jugée plus flexible par les développeurs, elle n'était pas compatible avec la plupart des serveurs des fournisseurs de services. Grâce à des différences conséquentes par rapport à WSDL 1.1, la version WSDL 2.0 est devenue une recommandation W3C depuis juin 2007. Dans la suite, je me focalise sur les principales différences entre les structures de description des deux versions les plus utilisées à savoir 1.1 et 2.0.

La version WSDL 1.1

La Figure 1.1 représente la grammaire qui organise la structure d'une description WSDL 1.1 [76]. L'élément englobant est *<definitions>*. Cette spécification est définie par six principaux éléments : *types*, *message*, *portType*, *binding*, *port* et *service*. Nous remarquons que cette syntaxe ne prévoit aucun élément ou attribut d'ordre sémantique pour la description.

Voici leurs définitions :

- *types* : définit les types de données utilisées dans les messages (lignes 3-5),
- *message* : représente une définition abstraite des données échangées ; un message est constitué de parties logiques, chacune est associée à un type déjà défini (lignes 7-9),
- *portType* : est constitué d'un ensemble de opérations abstraites ; chacune fait référence à un message en entrée "*input*" et/ou des messages de sortie "*output*" (lignes 11-17),
- *binding* : spécifie le protocole de transmission des données pour les opérations et les messages définis pour un *portType* en particulier (lignes 19-25),
- *port* : spécifie une adresse pour un *binding* (ligne 28),
- *service* : agrège un ensemble associé de *ports* (lignes 27-29).

WSDL 1.1 définit quatre primitives de transmission pour les *endpoint*s :

- *One-way* : l'*endpoint* reçoit un message,
- *Request-response* : l'*endpoint* reçoit un message et retourne un message en réponse,
- *Solicit-response* : l'*endpoint* envoie un message et reçoit un message en réponse,
- *Notification* : l'*endpoint* envoie un message.

```
1  <wsdl:definitions name="nmtoken"? targetNamespace="uri"?>
2   <import namespace="uri" location="uri"/>*
3   <wsdl:types> ?
4           <xsd:schema .... />*
5   </wsdl:types>
6
7   <wsdl:message name="nmtoken"> *
8           <part name="nmtoken" element="qname"? type="qname"?/> *
9   </wsdl:message>
10
11  <wsdl:portType name="nmtoken">*
12          <wsdl:operation name="nmtoken">*
13                  <wsdl:input name="nmtoken"? message="qname" />?
14                  <wsdl:output name="nmtoken"? message="qname" />?
15                  <wsdl:fault name="nmtoken" message="qname" /> *
16          </wsdl:operation>
17  </wsdl:portType>
18
19  <wsdl:binding name="nmtoken" type="qname">*
20          <wsdl:operation name="nmtoken">*
21                  <wsdl:input /> ?
22                  <wsdl:output /> ?
23                  <wsdl:fault name="nmtoken" /> *
24          </wsdl:operation>
25  </wsdl:binding>
26
27  <wsdl:service name="nmtoken"> *
28          <wsdl:port name="nmtoken" binding="qname" /> *
29  </wsdl:service>
30
31 </wsdl:definitions>
```

FIGURE 1.1 – Structure d'une description WSDL 1.1

La version WSDL 2.0

La Figure 1.2 représente la grammaire qui organise la structure d'une description WSDL 2.0 [13]. L'élément englobant dans la spécification WSDL 2.0 est <*description*> et est définie par cinq principaux éléments : : *types*, *interface*, *binding*, *endpoint* et *service*. Nous remarquons que cette syntaxe est plus riche que la précédente. Sa spécification est plus adaptée à l'extension si besoin d'ajout d'élément ou attribut d'ordre sémantique pour la description.

Voici leurs définitions :

- *types* : définit les types de données utilisées dans les messages (lignes 4-8),
- *interface* : décrit une séquence de messages qu'un service envoie et/ou reçoit. Elle assure ceci en regroupant les messages associés dans des opérations. Une opération est une séquence de "*input*" et de "*output*" (lignes 10-15). Une interface est un ensemble d'opérations (lignes 10-17).
- *binding* : spécifie le protocole de transmission des données pour les opérations et les messages définis pour une *interface* en particulier (lignes 17-25),
- *endpoint* : spécifie une adresse pour un *binding* (ligne 27),
- *service* : agrège un ensemble associé de *endpoint*s (lignes 26-28),

La différence entre cette version et la précédente se résume dans :

- La possibilité d'extension et d'ajout d'éléments et/ou d'attributs (pour l'ajout de la sémantique dans le langage de description). Cette extension a été exploitée par SAWSDL (voir section 1.2.2) et mon approche YASAWSDL (voir chapitre 4),
- La suppression des éléments XML "*message*",
- Le remplacement de l'élément XML "*portType*" par l'élément XML "*interface*",
- Le remplacement de l'attribut XML "*port*" par l'élément XML "*endpoint*".

```
1  <description targetNamespace="xs:anyURI" >
2                  [ <import namespace="xs:anyURI" location="xs:anyURI"? />
3                  | <include location="xs:anyURI" /> ]*
4          <types>
5                  [ <xs:import namespace="xs:anyURI" schemaLocation="xs:anyURI"? /> |
6                  <xs:schema targetNamespace="xs:anyURI"? /> |
7                  other extension elements ]*
8          </types>?
9   <interface>
10         <operation name="xs:NCName" pattern="xs:anyURI"? style="list of xs:anyURI"? >
11                 <input messageLabel="xs:NCName"? element="union of xs:QName, xs:token"? />
12                 <output messageLabel="xs:NCName"? element="union of xs:QName, xs:token"? />
13                 <infault ref="xs:QName" messageLabel="xs:NCName"? />*
14                 <outfault ref="xs:QName" messageLabel="xs:NCName"? />*
15         </operation>
16  </interface>
17  <binding name="xs:NCName" interface="xs:QName"? type="xs:anyURI" >
18         <fault ref="xs:QName" />
19         <operation ref="xs:QName" >
20                 <input messageLabel="xs:NCName"? />
21                 <output messageLabel="xs:NCName"? />
22                 <infault ref="xs:QName" messageLabel="xs:NCName"? />
23                 <outfault ref="xs:QName" messageLabel="xs:NCName"? />
24         </operation>
25  </binding>
26  <service name="xs:NCName" interface="xs:QName" >
27         <endpoint name="xs:NCName" binding="xs:QName" address="xs:anyURI"? />+
28  </service>
29 </description>
```

FIGURE 1.2 – Structure d'une description WSDL 2.0

Dans la suite, je propose un exemple de description de service Web que je reprendrai à chaque fois que j'illustrerai une approche de description de service. L'exemple est le suivant : Une agence immobilière met à la disposition des clients et de ses agents mobiles différents services Web pour consulter les différents types de biens, les localiser et s'y rendre s'ils veulent afin de les visiter.

L'exemple étudié ici est un cas d'utilisation : "Localiser en GPS un appartement pour l'achat et avoir un plan de carte pour s'y rendre".

Les opérations du service Web sont définies comme suit :

– "getLocationGPS" renvoie la position actuelle de la personne demandant le service Web en termes de latitude et de longitude en coordonnées GPS.
– "getApartmentBuyList" récupère la liste des appartements proches et disponibles à l'achat.
– "getMap" récupère le plan de carte et les informations nécessaires pour pouvoir aller à l'appartement.

La Figure 1.3 présente un extrait de la description WSDL 2.0 du service Web de l'agence immobilière. Ultérieurement, cet exemple d'extrait de description vous sera utile pour comprendre l'enrichissement et l'extension apporté par notre contribution au standard.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:description
  targetNamespace="http://www.telecom-sudparis.eu/GeoLocApartBuyMapService.wsdl20"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdl="http://www.w3.org/ns/wsdl"
  xmlns:tns="http://www.telecom-sudparis.eu/GeoLocApartBuyMapService/"
  name="GeoLocApartBuyMapWebService">

  <wsdl:types>
    <xs:import    namespace="http://www.telecom-sudparis.eu/GeoLocApartBuyMapService.xsd"
              schemaLocation="http://www.telecom-sudparis.eu/GeoLocApartBuyMapService.xsd" />
  </wsdl:types>

  <wsdl:interface   name="GeoLocApartBuyMapServiceInterface">

    <wsdl:operation   name="getLocationGPS">
      <wsdl:input   element="tns:getSignalGPS"/>
      <wsdl:output   element="tns:PositionGPS"/>
    </wsdl:operation>

    <wsdl:operation   name="getApartmentBuyList">
      <wsdl:input   element="tns:ApartmentData"/>
      <wsdl:output   element="tns:Apartment"/>
    </wsdl:operation>

    <wsdl:operation   name="getMap">
      <wsdl:input   element="tns:MapData"/>
      <wsdl:output   element="tns:Map"/>
    </wsdl:operation>
  </wsdl:interface>

  <wsdl:binding   name="GeoLocApartBuyMapServiceSOAPBinding"
    interface="tns:GeoLocApartBuyMapServiceInterface" ...>    ...  </wsdl:binding>

  <wsdl:service   name="GeoLocApartBuyMapWebService"
    interface="tns:GeoLocApartBuyMapServiceInterface">
    <wsdl:endpoint   name="GeoLocApartBuyMapServiceEndpoint"
      binding="tns:GeoLocApartBuyMapServiceSOAPBinding"
      address="http://www.telecom-sudparis.eu/GeoLocApartBuyMapWebService/" />
  </wsdl:service>
</wsdl:description>
```

FIGURE 1.3 – Extrait de la description WSDL du service Web de l'agence immobilière.

1.2.2 L'approche SAWSDL

Semantic annotation for *WSDL* (*SAWSDL*) [59] est un langage sémantique de description de service Web. Il est évolutif et compatible avec les standards des services Web existants, et plus spécifiquement avec *WSDL* [15]. *SAWSDL* augmente l'expressivité du langage *WSDL* avec la sémantique en utilisant des concepts analogues à ceux utilisés dans *OWL-S* [77]. D'une part *SAWSDL*, fournit un mécanisme permettant d'annoter sémantiquement les types de données, les opérations, les entrées et les sorties de *WSDL* et d'autre part, il ajoute des éléments pour spécifier les pré-conditions, les effets et les catégories des services Web. Les aspects relatifs à la qualité et l'orchestration des services ne sont pas traités dans *SAWSDL*.

Je présente dans ce qui suit les motivations qui étaient à l'origine de *SAWSDL*.

Motivations de SAWSDL

En plus de la découverte et l'invocation automatiques des services Web, déjà citées dans l'introduction, SAWSDL vise la réalisation des objectifs suivants :

- *Un langage au dessus des standards des services Web existants :* les standards de services Web sont devenus rapidement une technologie préférée pour l'intégration des applications. Les entreprises investissent dans des projets d'intégration basés sur des services Web. Par conséquent, les concepteurs de *SAWSDL* considèrent que n'importe quelle approche pour la description sémantique des services Web doit être compatible avec l'existant. A cet effet *SAWSDL*, une extension sémantique de *WSDL* a été conçue,
- *Concevoir un langage incrémental :* il est relativement facile de modifier les outils existants autour de *WSDL* afin qu'ils s'adaptent à *SAWSDL*. Ce qui fait de *SAWSDL* une approche incrémentale,
- *Le mécanisme d'annotation doit être indépendant du langage de représentation de la sémantique :* Il y a un certain nombre de langages potentiels pour représenter la sémantique comme Web Service Modeling Language (WSML) [19], *OWL* et Unified Modeling Language (UML) [75]. Chaque langage offre différents degrés d'expressivité. La position des concepteurs de *SAWSDL* est qu'il n'est pas nécessaire d'attacher les standards des services Web à un langage sémantique particulier. Cette approche est la vision décrite dans [86] et donne plus de flexibilité aux développeurs.

Dans ce qui suit, je présente comment l'annotation sémantique est faite dans *SAWSDL*.

Annotation sémantique

L'annotation sémantique des documents *WSDL* est possible grâce à l'extensibilité de *WSDL* 2.0. En effet, conceptuellement *WSDL 2.0* est doté des constructions suivantes : interface, opération, message, binding, service et endpoint. Les trois premiers à savoir interface, operation et message concernent la définition abstraite du service tandis que les trois autres sont relatifs à l'implémentation du service.

SAWSDL fournit des mécanismes pour référencer des concepts de modèles définis à l'extérieur du document *WSDL*. Cela se fait grâce à l'attribut "sawsdl". Il existe trois extensions de cet attribut. La première est *modelReference* et permet d'associer un composant *WSDL* ou *XML*

Schema à un concept d'une ontologie. Les deux autres sont *liftingSchemaMapping* et *loweringSchemaMapping* et permettent de spécifier la correspondance (dit *Mapping*) entre les données sémantiques et les éléments XML. Les *SchemaMapping* sont utilisés pour établir la correspondance entre les structures des entrées et des sorties, et est utile lorsque les structures *XML* demandées par le client et celles fournies par le service sont différentes.

L'annotation des interfaces, opérations, entrées/sorties et les types *XML* simples s'effectue en leur associant un concept dans une ontologie par le biais de l'attribut *modelReference*. Cependant, l'annotation des types de données *XML* complexes peut nécessiter en plus un Schema Mapping. En effet, deux services Web peuvent manipuler le même type complexe mais avec deux structures différentes. La Figure 1.4 présente un extrait de la description SAWSDL du service Web de l'agence immobilière. L'interface du service est annotée par trois concepts de l'ontologie "agensimo" (lignes 16-18). L'opération "getLocationGPS" est annotée par deux concepts (lignes 21-22). Les annotations de l'input "getSignalGPS" et l'output "PositionGPS" (lignes 24 et 26) précisent que l'opération prend en entrée un signal GPS et renvoie en sortie une position GPS. Ces annotations associent les éléments fonctionnels à des concepts tels que : *achat* et *appartement* ce qui décrit sémantiquement ce service et le distingue des autres.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <wsdl:description
3    targetNamespace="http://www.telecom-sudparis.eu/GeoLocApartBuyMapService.wsdl20"
4    xmlns="http://schemas.xmlsoap.org/wsdl/"
5    xmlns:wsdl="http://www.w3.org/ns/wsdl"
6    xmlns:tns="http://www.telecom-sudparis.eu/GeoLocApartBuyMapService/"
7    xmlns:sawsdl="http://www.w3.org/ns/sawsdl"
8    name="GeoLocApartBuyMapWebService">
9
10   <wsdl:types>
11     <xs:import    namespace="http://www.telecom-sudparis.eu/GeoLocApartBuyMapService.xsd"
12              schemaLocation="http://www.telecom-sudparis.eu/GeoLocApartBuyMapService.xsd" />
13   </wsdl:types>
14
15   <wsdl:interface name="GeoLocApartBuyMapServiceInterface"
16     sawsdl:modelReference="http://localhost/agensimo#terminalGPS
17                            http://localhost/agensimo#buy
18                            http://localhost/agensimo#apart">
19
20     <wsdl:operation name="getLocationGPS"
21       sawsdl:modelReference="http://localhost/agensimo#GPSactive
22                              http://localhost/agensimo#GPSposition">
23       <wsdl:input        element="tns:getSignalGPS"
24         sawsdl:modelReference="http://localhost/agensimo#GPSsignal"/>
25       <wsdl:output         element="tns:PositionGPS"
26         sawsdl:modelReference="http://localhost/agensimo#GPScoordinates"/>
27     </wsdl:operation>
28
29     <wsdl:operation   name="getApartmentBuyList">...</wsdl:operation>
30     <wsdl:operation   name="getMap">...</wsdl:operation>
31   </wsdl:interface>
32
33   <wsdl:binding   name="GeoLocApartBuyMapServiceSOAPBinding"
34     interface="tns:GeoLocApartBuyMapServiceInterface" ...>    ...  </wsdl:binding>
35
36   <wsdl:service   name="GeoLocApartBuyMapWebService"
37     interface="tns:GeoLocApartBuyMapServiceInterface">
38     <wsdl:endpoint   name="GeoLocApartBuyMapServiceEndpoint"
39       binding="tns:GeoLocApartBuyMapServiceSOAPBinding"
40       address="http://www.telecom-sudparis.eu/GeoLocApartBuyMapWebService/" />
41   </wsdl:service>
42 </wsdl:description>
```

FIGURE 1.4 – Extrait de la description SAWSDL du service Web de l'agence immobilière.

Je présente dans la section suivante une approche qui s'inspire aussi de WSDL : *USDL*.

1.2.3 L'approche USDL

Développé au sein du laboratoire de recherche ALPS (*Applied Logic Programming-Languages and Systems*) de l'Université du Texas, l'approche USDL [4] (*Universal Service-Semantics Description Language*) propose une description sémantique et formelle pour les services Web. Elle se base sur l'utilisation de l'ontologie OWL WordNet [92].

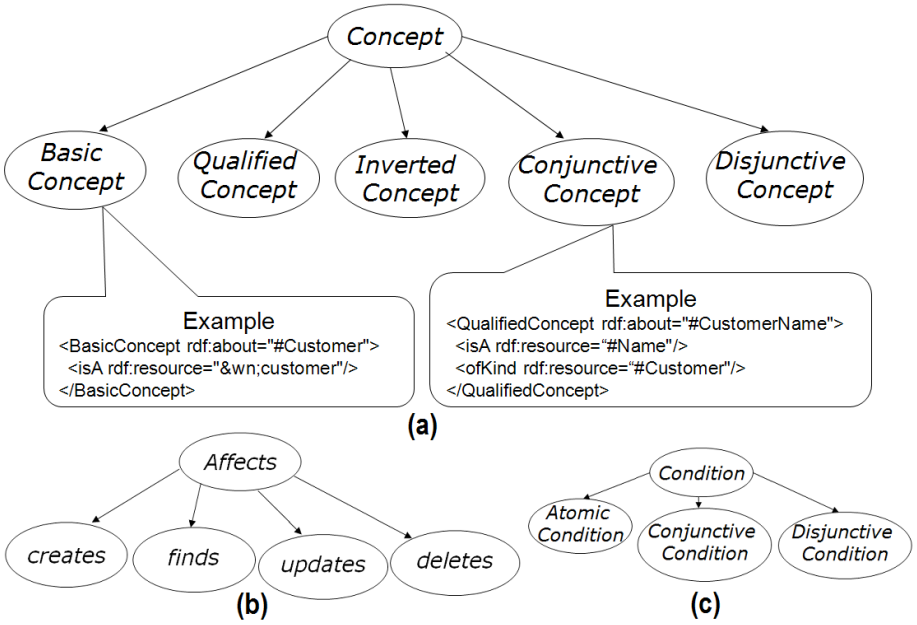


FIGURE 1.5 – Principaux éléments de description dans USDL [48].

Éléments de de base de l'approche USDL

Cette approche offre un langage qui décrit formellement et sémantiquement les services Web [85]. Pour ce faire, il considère que l'ontologie OWL WordNet est un outil universel grâce auquel tout le monde retrouve une représentation commune des concepts du monde réel. USDL propose une description basée sur des concepts en utilisant la logique de proposition. L'approche justifie cette motivation pour l'aspect formel et l'utilisation de WordNet par l'abondance de relations sémantiques entre les éventuels termes figurant dans cette ontologie et dont on peut profiter pour décrire les services Web. Elle propose de puiser dans les relations fournies par l'ontologie OWL WordNet, telles que l'hyponymie, l'antonymie, la synonymie, etc.

USDL propose une construction de description sémantique au-dessus de la description syntaxique WSDL 1.1. Cette construction engendre des modifications dans la syntaxe et la structure [48]. La Figure 1.5 illustre les trois principaux éléments pour décrire un service Web :

- La classe *Concept* : est une classe générique pour modéliser des concepts du monde réel,
- La propriété *Affects* : est une classe générique pour décrire des effets du service sur le monde réel,
- La classe *Conditions* : est une classe générique pour décrire des contraintes pour le service.

La Figure 1.5 illustre les sous-concepts de chacun des éléments de base d'une description USDL. La Figure 1.5.(a) illustre les sous-concepts de la classe *Concept* et donne deux extraits

de code comme exemples. La Figure 1.5.(b) illustre les sous-concept de la propriété *Affects*. La Figure 1.5.(c) illustre les sous-concept de la classe *Conditions*.

Construction d'une description USDL

Comme WSDL 1.1, USDL décrit un service Web en termes des balises XML "*Messages*" et "*PortType*". L'utilisation de l'ontologie WordNet intervient à ces niveaux. Pour décrire les opérations du service (au niveau des *PortType*) et leurs paramètres (au niveau des *Messages*), USDL propose de les associer à des concepts basiques, des concepts qualifiés, des concepts inversés, des concepts conjonctif et des concepts disjonctifs provenant de WordNet (voir Figure 1.5.(a)).

La sémantique spécifie comment l'environnement autour du service Web est affecté, autrement dit, l'effet de l'exécution du service Web sur le monde. USDL limite sa considération aux propriétés sémantiques des effets secondaires suite à une opération de création, de mise à jour, de suppression ou de recherche (voir Figure 1.5.(b)). Par ailleurs, si aucun de ces effets n'est appliqué, alors un effet générique est utilisé. Toute application qui aurait besoin de faire appel à un service USDL doit impérativement avoir la capacité de raisonner avec le lexique et les termes spécifiques de WordNet appelés atoms (tels que les *lemmas, senses, lexical semantic relations,*etc.)

En appliquant l'approche USDL sur un service Web écrit en WSDL, il sera formellement défini comme une fonction associée à un ou des effets, en spécifiant le tout en concepts provenant de WordNet.

La Figure 1.6 présente un extrait de la description USDL du service Web de l'agence immobilière. L'interface du service est annotée par trois concepts de l'ontologie "agensimo" (lignes 16-18). L'opération "getLocationGPS" est annotée par deux concepts (lignes 21-22). L'input "getSignalGPS" et l'output "PositionGPS" sont chacun annoté par un concept (lignes 24 et 26).

```
1  <definitions>
2    <PortType rdf:about="#GeoLocApartBuyMap_Service">
3      <hasOperation rdf:resource="#getLocationGPS" />
4    </PortType>
5
6    <Operation rdf:about="#getLocationGPS ">
7      <hasInput  rdf:resource="#ReserveFlight_Request"/>
8      <hasOutput rdf:resource="#ReserveFlight _Response"/>
9      <creates rdf:resource="#" />
10   </Operation>
11
12   <Message rdf:about="#getLocationGPS _Request">
13     <hasPart rdf:resource="#getSignalGPS" />
14   </Message>
15
16   <Message rdf:about="#getLocationGPS _Response">
17    <hasPart rdf:resource="#PositionGPS"/>
18   </Message>
19
20   <BasicConcept rdf:about="#getSignalGPS">
21     <isA rdf:resource="&wn;GPSsignal"/>
22   </BasicConcept>
23
24   <BasicConcept rdf:about="#PositionGPS">
25     <isA rdf:resource="&wn;GPScoordinates"/>
26   </BasicConcept>
27 ...
28 </definitions>
```

FIGURE 1.6 – Extrait de la description USDL du service Web de l'agence immobilière.

La description USDL garde pratiquement la même hiérarchie des éléments que WSDL mais apporte quelques modifications et de nouveaux éléments XML. Les détails de l'opération *getLocationGPS* sont exportés à l'extérieur de son *portType* (lignes 3 et 6). Les inputs et outputs sont définis sous forme d'éléments *Message* (lignes 12 et 16). Chaque *Message* pointe vers un élément *BasicConcept* (lignes 20 et 24) qui fait référence à un concept provenant de WordNet (lignes 21 et 25).

1.3 Approches basées sur des langages sémantiques

1.3.1 L'approche OWL-S

Semantic Markup for Web Services OWL-S [77, 17, 57] est un langage permettant de décrire les services Web de façon non ambiguë et interprétable par des programmes. Ce langage est basé sur le langage d'ontologie du Web (OWL) [2]. *OWL-S* est aussi une ontologie *OWL* particulière. Je présente dans la suite les motivations de *OWL-S* et son ontologie supérieure pour les services Web.

Motivations de OWL-S

OWL-S s'inscrit dans les approches de *la composition automatique des services Web*. L'accomplissement d'une tâche complexe implique la sélection, la composition automatique des services Web. Par exemple, l'utilisateur peut vouloir faire tout le nécessaire pour son voyage à une conférence. Il veut acheter un billet mais aussi il veut réserver un hôtel proche du lieu de la conférence. Il peut également ajouter des contraintes de coût minimal. Afin qu'un agent puisse exécuter ce type de tâche, *OWL-S* fournit une spécification des pré-requis et des conséquences de l'exécution de chaque service individuel ainsi qu'un langage pour décrire les services composés et le flux de données. Pour atteindre ces objectifs, *OWL-S* définit une ontologie supérieure pour la description, l'invocation et la composition des services Web. Celle-ci est présentée dans ce qui suit.

L'ontologie supérieure de OWL-S

La structuration de l'ontologie supérieure de *OWL-S* est motivée par la nécessité de fournir trois types d'information essentiels pour un service, à savoir :

- *Que réalise le service :* cette information est donnée dans le *Service Profile*. Ce dernier permet la description, la publication et la découverte des services, en spécifiant une description textuelle à destination des utilisateurs "humains", des propriétés fonctionnelles et des propriétés non fonctionnelles. Dans l'approche *OWL-S*, la section *Profile* est utilisée à la fois par les fournisseurs pour publier leurs services et par les clients pour spécifier leurs besoins. Par conséquent, elle constitue l'information utile pour la découverte et la composition de services [77]. Les recherches de services peuvent se baser sur n'importe quel élément de *Profile* comme critère [12].
- *Comment utilisons-nous le service :* cette information est fournie dans le *Service Model* qui est utilisé pour, entre autres, composer les services. *OWL-S* modélise les services en tant que processus et celui-ci est défini par ses entrées/sorties. Trois types de processus

existent : les processus atomiques (*AtomicProcess*), simples (*SimpleProcess*) et composites (*CompositeProcess*). Un processus atomique représente le niveau le plus fin pour un processus et correspond à une action que le service peut effectuer en une seule interaction avec le client. Les processus composites sont décomposables en d'autres processus (composés ou non) ; leur décomposition peut être spécifiée en utilisant un ensemble de structures de contrôles tels que : *Sequence*, *Split*, *If-Then-Else* etc. Un processus atomique est utilisé pour fournir une vue d'un processus atomique ou une représentation simplifiée d'un processus composite.

– *Comment accèdons-nous au service :* cette information est donnée dans le *Service Grounding*. Celui-ci indique comment accéder concrètement au service et fournit les détails concernant les protocoles, les formats de messages et les adresses physiques. Cette information est particulièrement utile pour l'invocation automatique de services.

La Figure 1.7 résume la structuration de l'ontologie supérieure de *OWL-S*.

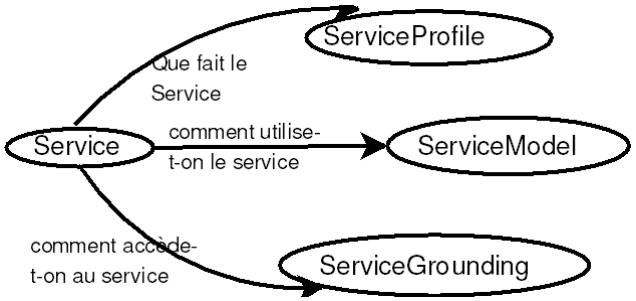


FIGURE 1.7 – L'ontologie supérieure de OWL-S

Je continue avec l'exemple du service Web de l'agence immobilière vu pour WSDL. La Figure 1.8 présente un premier extrait de la description *OWL-S*. Elle est composée du profil du service (*GeoLocApartBuyMapProfile*), du processus du service (*GeoLocApartBuyMapProcess*) et du *Grounding* du service (*GeoLocApartBuyMapGrounding*).

```
1 <service:Service rdf:ID="GeoLocApartBuyMapService">
2         <service:presents rdf:resource="&my_profile;#GeoLocApartBuyMapProfile"/>
3         <service:describedBy rdf:resource="&my_process;#GeoLocApartBuyMapProcess"/>
4         <service:supports rdf:resource="&my_grounding;#GeoLocApartBuyMapGrounding"/>
5 </service:Service>
```

FIGURE 1.8 – Service OWL-S du service Web de l'agence immobilière.

La Figure 1.9 présente le profil du service. Tout d'abord, un identifiant est affecté à ce profil (ligne 2). Ensuite, on donne le processus ainsi que le service auquel appartient ce profil (lignes 4-5). Les lignes 7-14 définissent un nom de service et donnent une description textuelle de ce service ainsi que les coordonnées et l'adresse de l'agence immobilière à contacter pour plus d'informations sur le service.

La Figure 1.10 présente le processus du service (*GeoLocApartBuyMapProcess*). Je l'ai simplifié pour plus de clarté ; je propose ici qu'il soit composé de trois processus atomiques *getLocationGPS*, *getApartmentBuyList* et *getMap* (lignes 17-25). *GeoLocApartBuyMapProcess* prend en entrée un signal GPS (*SignalGPS*, lignes 3-8) et fournit en sortie le plan de carte avec informations pour s'y rendre à l'appartement à vendre (*getMap*, lignes 10-15).

```
<profileHierarchy:GeoLocApartBuyMap
         rdf:ID="GeoLocApartBuyMapProfile">

        <service:presentedBy rdf:resource="&my_service;#GeoLocApartBuyMapService"/>
        <profile:has_process rdf:resource="&my_process;#GeoLocApartBuyMapProcess"/>

        <profile:serviceName>GeoLocApartBuyMapService</profile:serviceName>
        <profile:textDescription>
                C'est la description OWL du workflow modélisant
                     les détails du service Web GeoLocApartBuyMap.
        </profile:textDescription>
        <profile:contactInformation>
                Informations pour contacter l'agence.
        </profile:contactInformation>

</profileHierarchy:GeoLocApartBuyMap>
```

FIGURE 1.9 – Le profil du service Web de l'agence immobilière.

Les deux types *SignalGPSInputType* et *MapOutputType* sont des classes OWL définissant respectivement le type d'un signal GPS et celui d'un plan de carte (position par rapport à l'appartement, adresse, bâtiment, étage, prix, dates de l'annonce, contacts, etc).

```
<process:CompositeProcess rdf:ID="GeoLocApartBuyMapProcess">

        <process:hasInput>
                <process:Input rdf:ID="SignalGPS">
                <process:parameterType rdf:datatype="&xsd;#anyURI">
                &THIS;SignalGPSInputType</process:parameterType>
                </process:Input>
        </process:hasInput>

         <process:hasOutput>
                 <process:Output rdf:ID="Map">
                         <process:parameterType rdf:datatype="&xsd;#anyURI">
                         &THIS;MapOutputType</process:parameterType>
                 </process:Output>
         </process:hasOutput>

         <process:composedOf>
                 <process:Sequence>
                         <process:components>
                                <process:Perform rdf:ID="getLocationGPS"/>
                                <process:Perform rdf:ID="getApartmentBuyList"/>
                                <process:Perform rdf:ID="getMap"/>
                         </process:components>
                 </process:Sequence>
        </process:composedOf>

</process:CompositeProcess>
```

FIGURE 1.10 – Le processus du service Web de l'agence immobilière.

1.3.2 L'approche WSMO

WSMO est une ontologie qui décrit les différents aspects relatifs à la composition dynamique des services Web, y compris la découverte dynamique, la sélection, la médiation et l'invocation. Elle est basée sur WSMF[25, 57] (Web Service Modelling Framework) qui spécifie les éléments principaux pour décrire les services Web sémantiques. De tels éléments incluent ontologies, objectifs, services Web et médiateurs. Les ontologies définissent la terminologie, utilisée par les autres éléments, en termes de concepts, relations, fonctions, instances et axiomes. Les buts

indiquent ce que l'utilisateur attend du service. La description du service Web définit les fonctionnalités offertes par le service. Les médiateurs lient les différents éléments afin de permettre l'interopérabilité entre les composants hétérogènes.

Le langage *WSML* [19] est utilisé pour décrire formellement tous les éléments de *WSMO* et l'environnement d'exécution *WSMX* [16] permet la découverte, la sélection, la médiation, l'invocation et l'interopérabilité des services Web sémantiques. Je présente dans la suite les motivations de *WSMO*.

Motivations de WSMO

WSMO [21] partage avec *OWL-S* les mêmes motivations à savoir la découverte, l'invocation et la composition automatiques des services Web. Cependant *WSMO* ajoute à celles-ci l'objectif suivant : *Un découplage fort entre les composants et un rôle central pour la médiation*. L'un des principes fondamentaux de *WSMO* consiste en la séparation totale entre les différents éléments impliqués dans la composition des services Web. A cet effet, *WSMO* veut distinguer comment le client formule sa demande et comment le fournisseur expose son service. Sachant que la demande et l'offre sont décrites de façons différentes, un travail important doit être effectué afin d'établir la correspondance entre la demande et l'offre. Ce travail est le rôle des médiateurs.

Facettes de WSMO

Les facettes de *WSMO* sont les *ontologies*, les *médiateurs*, les *services Web* et les *objectifs*. Les ontologies fournissent la sémantique compréhensible par une machine pour les informations utilisées par tous les acteurs d'un service Web. *WSMO* permet d'importer une ontologie dans une autre ontologie soit directement, quand il n'y pas de conflits entre les concepts, soit indirectement, par le biais d'un médiateur qui va résoudre les conflits possibles. Les blocs de base d'une ontologie sont concepts, relations, fonctions, instances et axiomes. Un ensemble de propriétés non fonctionnelles est donné généralement au début de chaque définition d'une ontologie. Les médiateurs permettent de lier différentes ressources hétérogènes et résoudre les incompatibilités à plusieurs niveaux. Il peut y avoir une incompatibilité entre les données ou entre les processus. Dans le premier cas, la médiation sert à établir la correspondance entre les différentes terminologies. Le deuxième type d'incompatibilité apparaît au moment de la communication entre différents services Web hétérogènes et dans ce cas, le médiateur fournit la fonctionnalité pour une analyse d'exécution de deux services Web donnés, et compense les éventuelles disparités.

Les services Web *WSMO* sont composés d'un ensemble optionnel de propriétés non fonctionnelles, un mécanisme permettant d'importer des ontologies soit directement en utilisant *importsOntology*, soit indirectement via un médiateur, une capacité décrivant la fonctionnalité du service en termes de pré et postconditions et effets et une interface décrivant le comportement du service vis-à-vis de ses partenaires.

Pour montrer comment le service *réservation* peut être décrit en *WSMO*, je commence par définir les concepts que ce service manipule. La Figure 1.11 présente la définition du concept *BuyApartmentOffer*. Tout d'abord, le nom du concept est défini (ligne 1). Ensuite, on définit les types de données présentées dans une offre de vente d'un appartement. La première donnée est de type *agent* et représente l'agent qui effectue la vente (ligne 2). La deuxième donnée est le prix de l'appartement (ligne 3). La troisième donnée est la date de mise en ligne de l'annonce (ligne 4). Enfin, la quatrième donnée est l'adresse de l'appartement (ligne 5).

```
concept BuyApartmentOffer
        vendeur impliesType agent
        prix ofType _float
        dateAnnonce ofType _date
        adresse ofType _string
```

FIGURE 1.11 – Description du concept BuyApartment en *WSMO*

```
instance BuyApartmentAdParis memeberOf BuyApartmentOffers
        vendeur CharlotteMontblanc
        prix 200.000
        dateAnnonce 20/02/2011
        adresse "20 rue Louis14, Paris 75014"
        detailsAppartement "F3, 66m2, 2e étage"
```

FIGURE 1.12 – Description d'une instance d'annonce d'un appartement à vendre en *WSMO*

Après avoir défini le concept *BuyApartmentOffer*, je crée une instance de *BuyApartmentAd-Paris*. La Figure 1.12 présente la définition de l'instance du concept *BuyApartmentOffer*. Ensuite, je donne dans la Figure 1.13 la définition du service permettant d'envoyer un plan de carte de l'appartement de l'offre à un client. Le service est accessible à l'adresse *http ://exempleBuyA-partmentOffers.org/GeoLocApartBuyMap* (ligne 1) et a la capacité *geoLocApartBuyMapCapacité* (ligne 2).

```
1 webServices _"http://exempleBuyApartmentOffers.org/GeoLocApartBuyMap"
2         capability geoLocApartBuyMapCapacité
```

FIGURE 1.13 – Description en *WSMO* du service *GeoLocApartBuyMap*

Enfin, je présente la définition de la capacité du service dans la Figure 1.14. Étant donné que la définition des *pré-conditions*, *post-conditions*, *hypothèses de base* et *effets* se fait de la même manière, nous nous contentons de donner celle des pré-conditions.

```
1  capability geoLocApartBuyMapCapacité
2          sharedVariable ?BuyApartmentOffers
3                  precondition
4          nonFonctionalProperties
5               #description hasValues "L'appartement est disponible pour l'achat
6                et son adresse est proche de la position actuelle du client."
7          endNonFonctionalProperties
8          definedBy
9                  #ApartToBuy(?BuyApartmentOffer.type.offre)
10         and  wsml#positionNearTo(?Apartment.address, wsml#currentPosition)
```

FIGURE 1.14 – Description en *WSMO* des conditions autour d'une capacité du service Web.

La pré-condition du service correspond au fait que la position du client soit proche de l'adresse de l'appartement et que l'appartement soit disponible pour l'achat. Tout d'abord la variable utilisée dans le service est définie (ligne 2). Ensuite, on donne la définition de la pré-condition du service. Cette définition commence par les propriétés non fonctionnelles du service (lignes 4-7). Ici, il s'agit d'une description textuelle. Enfin, je définis les deux conditions (ligne 8-10).

1.4 Examen des approches de description existantes

Dans cette section, j'examine les divers travaux présentés dans l'état de l'art de description de services Web. Pour ce faire, je commence par identifier les principaux critères de comparaison et j'utilise ces critères pour évaluer les approches étudiées.

1.4.1 Identification des critères d'évaluation

Dans mes travaux, je me fixe un ensemble d'objectifs qui guident ma contribution en termes de description sémantique de services Web.

Que ce soit en proposant tout un langage pour décrire les services et leur sémantique, ou en annotant les modèles de description existants, certaines approches dépendent d'un langage de description d'ontologie en particulier. Mon objectif est de proposer une description de services Web qui ne dépend pas d'un langage de description d'ontologie en particulier pour décrire l'aspect sémantique du service Web. J'identifie ainsi un premier critère : la dépendance vis-à-vis d'un langage de description d'ontologie.

Par ailleurs, il existe deux classes d'approches de description de services Web sémantiques. Celle qui développe un langage décrivant en même temps les services Web et leur sémantique, et celle qui annote des modèles de descriptions existants avec des données sémantiques. Cette dernière démarche facilite l'adaptation des descriptions déjà développées par les fournisseurs aux solutions proposées. Mon objectif est de proposer une approche de description qui adapte facilement les mécanismes standards de description et réutilise les outils déjà disponibles. J'identifie ainsi un deuxième critère : La réutilisation et l'adaptation de descriptions et d'outils de services Web

Il est clair que selon le principe de description choisi, l'expressivité ne sera pas équivalente pour les différentes approches. Cela dépendra du nombre et de l'importance des éléments fonctionnels décrits et de la capacité d'intégrer des informations non fonctionnelles. Mon objectif est de proposer une description de service qui décrit la fonctionnalité avec le plus possible d'éléments tout en ayant la capacité d'intégrer de nouvelles informations non-fonctionnelles et fonctionnelles ultérieurement. J'identifie ainsi un troisième critère : l'expressivité.

L'aspect métier des services Web est utilisé par quelques approches pour décrire sémantiquement les services. Mais parfois certaines approches soit sous-entendent cet aspect et elles s'arrêtent aux annotations sémantiques de domaine, soit ne sont pas assez explicites sur les concepts métiers ciblés par l'annotation sémantique. Il est clair donc que selon l'approche d'annotation choisie, la sémantique ne serait pas autant explicite et précise pour les différentes approches. Dans certaines approches quand les informations sémantiques décrivent des conditions, elles n'explicitent pas quels types de conditions pourtant il est très différents, par exemple, de les interpréter comme des pré-conditions ou comme des post-conditions. Dans d'autres approches, on n'explicite même pas que des annotations soient des effets ou des conditions et ceci relève de l'ambiguïté. Cela dépendra de la capacité des annotations à préciser l'information sémantique qu'on leur associe et les éléments fonctionnels qu'elles couvrent. Mon objectif est de proposer une description sémantique aussi explicite que précise et qui couvre des éléments fonctionnels importants. Bien que sur ce point la réflexion semble encore relever de l'expressivité, mais le caractère plus approfondi et pointé sur un critère de formulation claire et non ambiguë de l'expressivité elle-même me mène à identifier un quatrième critère : l'explicitation de la sémantique.

Les critères identifiés sont utilisés dans la suite pour examiner les approches de descriptions de services Web étudiées :

1. Dépendance vis-à-vis d'un langage d'ontologie
2. Réutilisation et adaptation de descriptions et d'outils
3. Expressivité
4. Explicitation de la sémantique

Pour la suite, la Table 1.1 présente la sémantique de symboles utilisée pour évaluer les approches de description par rapport aux critères identifiés ci-dessus.

TABLE 1.1 – Sémantique des symboles utilisés dans l'examen des approches de description.

Symboles	-	+/-	+	++
Interprétations	Ne répond pas au critère	Il y a un manque	Répond mais on peut améliorer	Répond bien au critère

1.4.2 Dépendance vis-à-vis d'un langage d'ontologie

Étant proche de *WSDL*, *SAWSDL* ne nécessite pas beaucoup d'efforts pour les développeurs habitués à WSDL contrairement à *OWL-S* et *WSMO*. *SAWSDL* permet d'utiliser tous types d'ontologies (OWL, WSML et UML) tandis que *OWL-S* accepte des ontologies *OWL* et *WSMO* des ontologies *WSML*. Sur ce point, l'approche USDL possède un inconvénient. Elle utilise exclusivement l'ontologie *OWL WordNet* et cela la classifie dans le rang des approches fermées. Elle ne propose pas encore de mécanismes alternatifs permettant d'intégrer et d'utiliser des ontologies spécifiques de domaine.

La Table 1.2 résume l'examen des approches de description de services Web selon ce critère. SAWSDL est indépendant du langage d'ontologie utilisé et les autres dépendent d'un langage d'ontologie en particulier.

TABLE 1.2 – Dépendances des approches vis-à-vis d'un langage d'ontologie.

Critère/Approches	Objectifs	SAWSDL	OWL-S	WSMO	USDL
Dépendance d'un langage d'ontologie	++ (Indépendant)	++	-	-	-

1.4.3 Réutilisation et adaptation de descriptions et d'outils

La réutilisation et l'adaptation sont souvent liées. Une facilité d'adaptation par rapport à une approche peut nous permettre de réutiliser les outils qu'on a utilisés avec l'approche. Même s'il y aura besoin d'adaptation des outils, cela sera minime et à faible coût.

Le critère d'adaptation concerne l'extensibilité de la description, de ses mécanismes d'annotation et des outils utilisés pour construire les descriptions, des outils d'annotation sémantique et/ou des outils d'analyse de descriptions de services (par exemple pour le parsing ou l'appariement).

En examinant les travaux selon le critère d'adaptation, j'ai remarqué qu'il existe deux classes d'approches de description de services Web sémantiques. Soit on développe tout un langage décrivant à la fois les services Web et leur sémantique dans une même structure tel que dans WSMO et OWL-S, soit on annote des modèles de descriptions existants avec des données sémantiques tels que dans SAWSDL. Cette dernière démarche facilite l'adaptation des descriptions déjà développées par les fournisseurs aux annotations et aux solutions proposées.

En termes de réutilisation et d'outillage, *OWL-S* tire partie de son ancienneté et ainsi il dispose de plusieurs outils allant du simple éditeur au composeur semi-automatique en passant par les outils d'appariement et de validation. Cependant *WSMO* ne dispose que des outils pour l'édition, *WSML Editor* et un seul outil propre à *SAWSDL* existe, *SAWSDL Tool Annotation* (un outil pour l'annotation sémantique). Les outils pour *WSMO* s'avèrent plus difficiles à développer car celui-ci se base sur *WSML*, un langage qui n'a pas été utilisé auparavant, tandis que *OWL-S* et *SAWSDL* s'appuient sur *RDF* [1] et *XML*. L'approche USDL nécessite la normalisation alors que celle-ci reste un processus lent surtout pour les ontologies spécifiques à l'industrie. Elle nécessite plusieurs itérations de conception pour aboutir à une version plus au moins stable dans tel ou tel secteur. D'autre part, on a toujours besoin d'utilisateurs qui comprennent les services et leurs sémantiques exactes pour réaliser leurs descriptions. La construction d'une description USDL fait intervenir beaucoup d'éléments et de codes descriptifs vu qu'elle est très verbeuse, par exemple par rapport à WSDL.

Prenons comme exemple la description de l'opération *getLocationGPS* dans l'extrait de description du service Web de l'agence immobilière. En USDL, pour décrire l'opération et l'interface englobante, on passe de 5 lignes (Figure 1.3) à plus de 20 lignes (Figure 1.6). D'ailleurs, même si on simplifie et on regarde un fichier USDL comme un fichier WSDL un peu modifié et complètement annoté, cela reste un travail lourd et coûteux surtout qu'il n'existe aucun outil automatique voire semi-automatique d'annotation sémantique USDL. La Table 1.3 résume l'examen des approches de description de services Web selon ce critère.

TABLE 1.3 – Facilité de réutilisation et d'adaptation pour les approches.

Critère/Approches	Objectifs	SAWSDL	OWL-S	WSMO	USDL
Adaptation & Réutilisation	++ (Standard,Outils)	++	+	+	+/-

1.4.4 Expressivité

Pour examiner l'expressivité, il faut étudier de plus près les moyens offerts par les approches afin de décrire les services Web en termes : sémantique, fonctionnel voire même non-fonctionnel, cela permet de vérifier jusqu'à quels niveaux les approches ont été expressives et s'il y a possibilité de l'être encore plus. Les ontologies sont utilisées pour décrire sémantiquement les services.

D'une part, les éléments décrits varient d'une approche à l'autre. Il est vrai qu'il peut avoir des éléments communs mais souvent chaque approche est plus expressive sur certains aspects que d'autres qui les considèrent pas et vice-versa. D'autre part, ce ne sont pas tous les éléments d'une description sémantique de service qui sont annotés sémantiquement. Ce point est très important dans l'expressivité sémantique d'une approche. La table de la Figure 1.15 résume les éléments de services Web spécifiés (que ce soit annotés ou non). Mon objectif est de proposer

une approche de description de services Web qui prend en charge tous les éléments mentionnés dans la table plus d'autres.

WSDL et SAWSDL spécifient quasiment les mêmes éléments avec une ou deux différences dans l'appellation. Dans SAWSDL, les annotations sémantiques concernent les parties interface, opération, input et output. Bien que OWL-S et WSMO spécifient plus d'éléments que SAWSDL, les annotations sémantiques ne couvrent pas la totalité des éléments. Selon la ligne de OWL-S dans la Figure 1.15, cette approche associe plusieurs éléments descriptifs au service lui même (modèle, version, profil, actor, etc) alors que WSMO ne lui associe qu'un élément dit "Goal" (Objectif) du service. Les deux approches OWL-S et WSMO ont des cases communes sur la partie "opérations" telles que : input, output, "precondition" et "effect", mais beaucoup plus d'éléments propres à chacun d'elles, tels que : "postcondition" et "assumption" pour WSMO, et "result" et "process" pour OWL-S.

La Figure 1.15 montre que l'approche USDL spécifie plus d'élément que WSDL et SAWSDL et qu'elle ajoute des éléments provenant de OWL-S tels que : "category", "effect" et "condition". Enfin, la nature de description de *SAWSDL* est purement fonctionnelle, il ne permet pas de définir des propriétés non fonctionnelles. La Table 1.4 résume l'examen des approches de description de services Web selon ce critère.

TABLE 1.4 – Expressivité des approches.

Critère/Approches	Objectifs	SAWSDL	OWL-S	WSMO	USDL
Expressivité	++	+	+/-	+/-	+/-

1.4.5 Explicitation de la sémantique des services

Il convient de rappeler qu'aucune parmi les approches examinées n'a atteint les objectifs initialement escomptés. En les évaluant par rapport aux objectifs communs : découverte et invocation automatiques, j'observe que ces objectifs sont partiellement pris en compte par *OWL-S*. D'une part, la description proposée par *OWL-S* a permis une découverte basée sur la sémantique qui est faite grâce à des outils, implémentant des algorithmes d'appariement, tel que OWLS-Matcher [70] mais d'autre part l'invocation automatique est réalisée en utilisant *WSDL*. *WSMO* et *SAWSDL* permettent uniquement l'invocation automatique. L'approche USDL affiche aussi des problèmes dûs à une contrainte semblable à la polysémie : il existe de nombreuses représentations différentes et distinctes en termes de sémantique formelle OWL, mais égales dans les termes syntaxiques des concepts réels qu'elles représentent. Ce qui peut prêter à ambiguïté si on les utilise en même temps (par exemple, comment distinguer le concept du terme tour (le monument ou le circuit touristique ?). S'agissant de *SAWSDL*, celui-ci s'intéresse à la découverte et l'invocation automatique des services mais il ne s'occupe pas de la composition. La Table 1.5 résume l'examen des approches de description de services Web selon ce critère.

TABLE 1.5 – Explicitation de la sémantique dans les approches.

Critère/Approches	Objectifs	SAWSDL	OWL-S	WSMO	USDL
Sémantique explicite	++	-	+	+	+/-

1.4.6 Récapitulatif

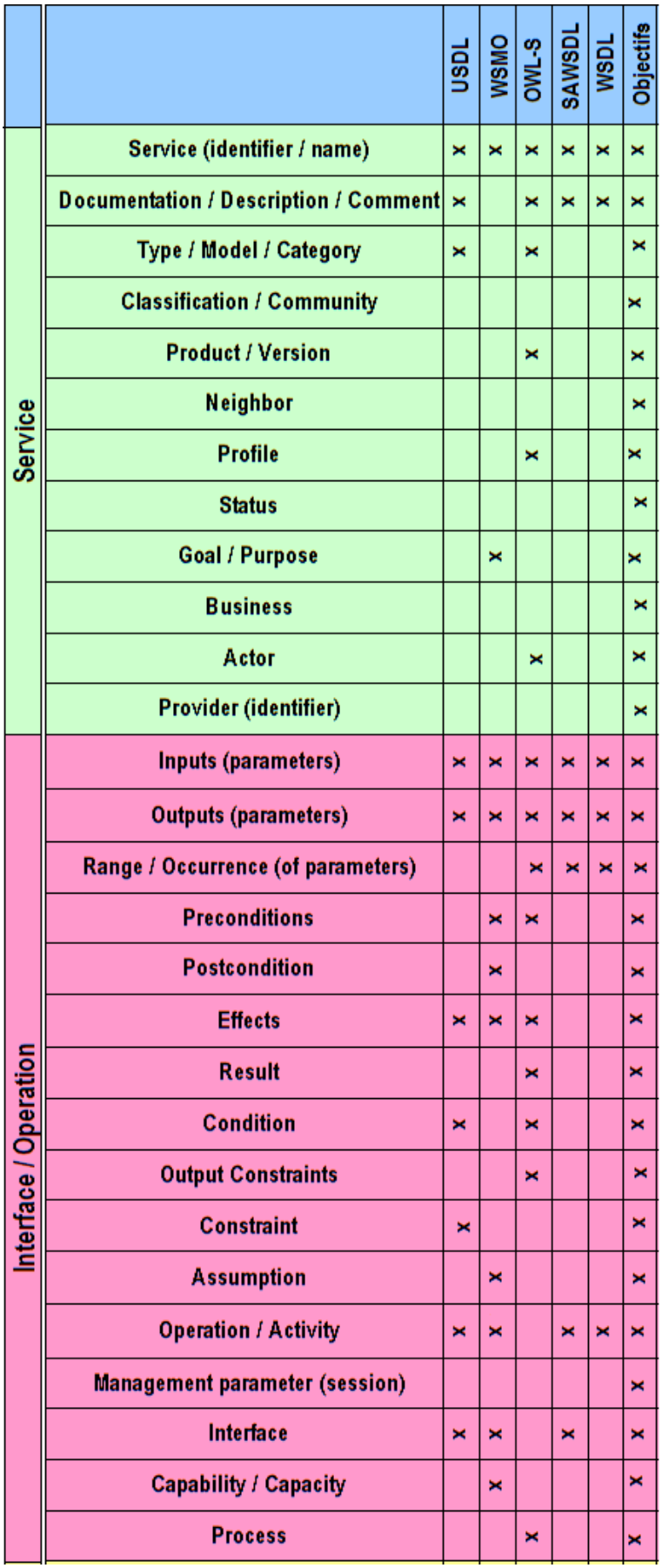
La Table 1.6 résume l'examen des approches de description de services Web selon les différents critères. Selon cette table, aucune des approches examinées ne réalise complètement mes objectifs. Toutefois, je peux m'inspirer de chacune des approches afin de combler les manques et proposer une approche qui correspond à mes motivations.

TABLE 1.6 – Table récapitulative de l'examen des approches de description de services Web.

Critère/Approches	Objectifs	SAWSDL	OWL-S	WSMO	USDL
Dépendance d'un type d'ontologie	++ (Indépendant)	++	-	-	-
Adaptation & Réutilisation	++ (Standard,Outils)	++	+	+	-
Expressivité	++ (Maximale)	+	+/-	+/-	+/-
Sémantique explicite	++ (Explicite)	-	+	+	+/-

1.5 Conclusion

Dans ce chapitre, j'ai présenté l'état de l'art de la description de services Web. Ensuite, j'ai examiné les différents travaux présentés. Pour examiner ces approches, j'ai commencé par identifier des objectifs par rapport à une contribution en description de services Web. Ces critères ont permis d'examiner les approches par rapport aux spécificités et l'expressivité de leurs langages de description et de leurs annotations sémantiques. Finalement, j'ai utilisé ces critères pour évaluer et comparer les approches étudiés. La discussion autour de l'état de l'art de la découverte et d'appariement de services a démontré le besoin d'une description indépendante de tout langage d'ontologie, plus facile en adaptation et en réutilisation, avec d'avantage d'expressivité et d'explicitation sémantique fonctionnelle et technique provenant du domaine des services Web.



		USDL	WSMO	OWL-S	SAWSDL	WSDL	Objectifs
Service	Service (identifier / name)	x	x	x	x	x	x
	Documentation / Description / Comment	x		x	x	x	x
	Type / Model / Category	x		x			x
	Classification / Community						x
	Product / Version			x			x
	Neighbor						x
	Profile			x			x
	Status						x
	Goal / Purpose		x				x
	Business						x
	Actor			x			x
	Provider (identifier)						x
Interface / Operation	Inputs (parameters)	x	x	x	x	x	x
	Outputs (parameters)	x	x	x	x	x	x
	Range / Occurrence (of parameters)			x	x	x	x
	Preconditions		x	x			x
	Postcondition		x				x
	Effects	x	x	x			x
	Result			x			x
	Condition	x		x			x
	Output Constraints			x			x
	Constraint	x					x
	Assumption		x				x
	Operation / Activity	x	x		x	x	x
	Management parameter (session)						x
	Interface	x	x		x		x
	Capability / Capacity		x				x
	Process			x			x

FIGURE 1.15 – Récapitulatif des éléments de description dans les approches étudiées

Chapitre 2

Stockage de descriptions de services

Sommaire

2.1 Introduction

L'intérêt des services Web dépend autant de la description que des mécanismes de stockage et de récupération de ces descriptions. Les premières approches de stockage de descriptions proposent des annuaires basés sur la norme UDDI qui mettent en relation dynamiquement un client à l'implémentation du service requis. Avec l'émergence des services Web sémantiques, de tels annuaires ne peuvent qu'assurer un support de base, ainsi des mécanismes additionnels doivent enrichir ou étendre tous les types d'annuaires (UDDI ou autres) pour mieux les adapter

aux architectures orientées service. Ces architectures sont porteuses de plus en plus d'informations bien évidemment fonctionnelles, mais en plus non-fonctionnelles ou sémantiques.

Dans la littérature, la réponse à des exigences d'extensibilité peut être apportée par deux principales approches : enrichir les structures de données des annuaires par des informations additionnelles et/ou étendre les structures actuelles des annuaires afin d'intégrer les nouveaux aspects. Souvent, les choix faits à ce niveau auront un grand impact sur les performances au moment de la publication et l'efficacité dans la phase de découverte des services. Par ailleurs, les annuaires de services de nos jours ne sont plus restreints à des environnements fermés d'entreprises, on cherche à les rendre disponibles à grande échelle et sur le Web (par exemple pour le domaine de la téléphonie mobile). Ceci permettra de les mettre au service des utilisateurs et des technologies qui ont besoin que les annuaires soit accessible au grand public du Web.

Après avoir présenté, dans le chapitre 1 les approches et les modèles de description de services, ce chapitre présente les différentes approches et modèles ainsi que les plateformes les plus connues de stockage de description de services. Je commence dans la section 2.2 par introduire le concept d'annuaires de services. La section 2.3 présente, au début, les standards pour la description de structure des annuaires de services, ensuite elle cite des approches basées sur ces standards. La section 2.4 présente les plateformes de stockage et recherche publiques sous forme de portails annuaires de services Web. Finalement, une synthèse de ce chapitre est présentée en section 2.5.

2.2 Les annuaires de services Web

Dans cette section, nous présentons les deux formes les plus répandues de plateformes de stockage de services Web à travers le concept d'annuaires des services et le concept de portail-annuaire de services Web.

2.2.1 Le concept d'annuaire de services

Un annuaire est une structure de gestion de services (publication, localisation, découverte). Sous forme de répertoire, il permet de stocker les informations nécessaires pour retrouver et accéder à un service, telles que les informations techniques et l'adresse des services Web, le nom de la personne/société qui gère un service donné, la description des fonctionnalités, etc.

2.2.2 Le concept de portail-annuaire de services Web

Les portails annuaires de services Web sont des annuaires Web publics qui se consacrent au stockage et à la découverte des services Web. Ces portails spécialisés hébergeant des services Web publics soit en utilisant des moteurs d'exploration ciblée ou en se basant sur l'enregistrement manuel. Ils offrent essentiellement des fonctionnalités de publication et de recherche via une interface Web. Certains portails offrent aussi une fonctionnalité de navigation. Ils s'appuient plutôt sur l'enregistrement manuel des services par les prestataires de services ou les utilisateurs du portail.

2.3 Annuaires basés sur des standards

OASIS [1] a défini, pour les annuaires de services, deux principales spécifications : UDDI et ebXML. Ces deux spécifications définissent des normes pour un annuaire de services d'usage général. Je commence par présenter les deux normes UDDI [73] et ebXML [72], ensuite, je présente des annuaires sémantiques de services Web basés sur ces standards.

2.3.1 Standards

Dans cette sous section, je présente les deux normes UDDI et ebXML.

UDDI

UDDI (Universal Description, Discovery and Integration) [73] est un standard défini par OASIS. Il définit la structure d'un annuaire de services Web.

Un service d'annuaire UDDI est un service Web qui gère les méta-données des services, l'information sur les fournisseurs de services et les implémentations des services. Afin de trouver un service Web, il est possible d'utiliser un annuaire UDDI en précisant des exigences concernant le service requis. On cherche le service par son nom et/ou par des mots-clés. Mais, souvent on a besoin de beaucoup de temps pour sélectionner, parmi les résultats retournés par l'annuaire, le service approprié. Plusieurs noms de services peuvent se ressembler et les mots-clés sont souvent associés à plusieurs services.

Les informations sur un service publié dans un annuaire UDDI se présentent sous trois catégories :

- Pages Blanches : adresse, contact, et identifiants des fournisseurs de services,
- Pages Jaunes : catégorisations industrielles fondées sur des normes de taxonomies,
- Pages Vertes : informations techniques sur les services publiés par les entreprises.

Avant que la norme UDDI soit créée en août 2000, les entreprises n'avaient pas une approche commune pour publier des informations sur leurs produits et leurs services Web pour leurs clients et leurs partenaires.

Un annuaire UDDI est conçu pour être interrogé par des messages SOAP et afin de pouvoir stocker et fournir des informations permettant l'accès aux documents WSDL (*Web Services Description Language*) décrivant les protocoles et les formats de messages nécessaires pour interagir avec les services Web répertoriés dans l'annuaire (voir Figure 2.1 [2]).

Les outils de recherche disponibles sont basés sur des mots-clefs et ne prennent pas en considération les relations entre les services Web et les caractéristiques sémantiques de chaque service Web, forçant l'utilisateur à recommencer la recherche depuis le début en utilisant de nouveaux termes clés.

Les spécifications UDDI inclue notamment :

- des API SOAP qui permettent l'interrogation et la publication d'informations,
- la représentation XML du modèle de données de l'annuaire et des formats de message SOAP,

1. OASIS : Organization for the Advancement of Structured Information Standards (www.oasis-open.org)
2. wiki.nectec.or.th/bu/ITM532Students_2008/WebService

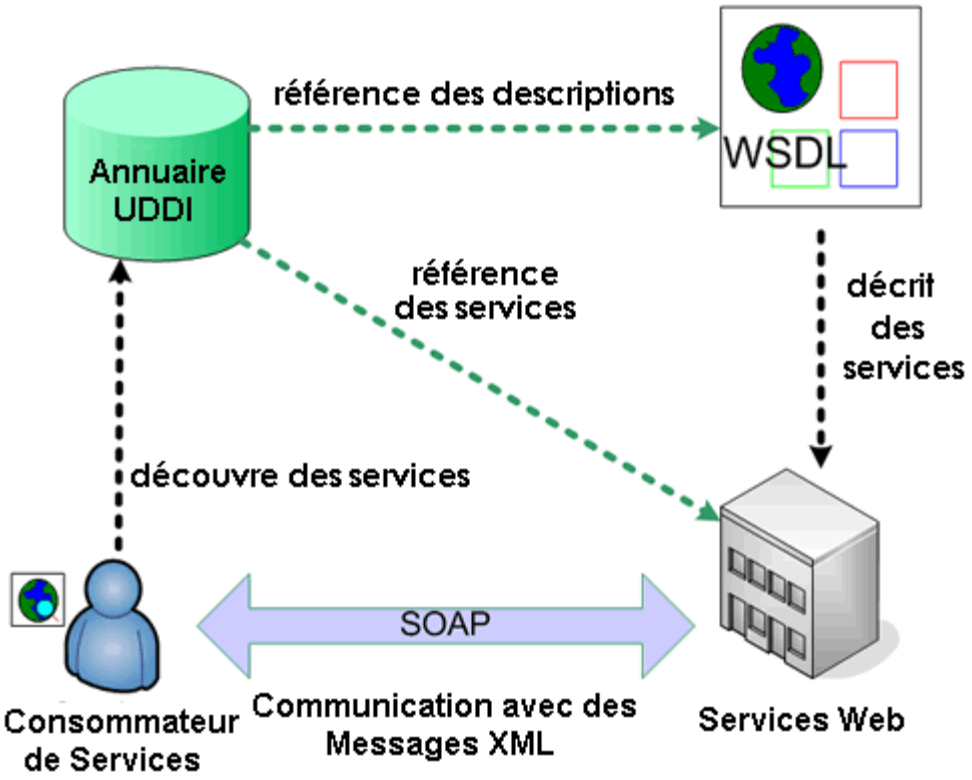


FIGURE 2.1 – Architecture basée sur un annuaire UDDI.

– des définitions de l'interface WSDL de SOAP,
– des définitions d'APIs de différents modèles techniques qui facilitent le travail des systèmes d'identification et de catégorisation des enregistrements UDDI.

ebXML

D'après la spécification définie par OASIS, un annuaire ebXML [72] est un système d'information qui gère en toute sécurité tout type de contenu et les méta-données normalisées qu'il décrit. *"Il fournit un ensemble de services qui permettent le partage de contenus et de méta-données entre les entités organisationnelles dans un environnement fédéré. Un annuaire ebXML peut être déployé dans un serveur d'applications, un serveur Web ou un autre conteneur de services. L'annuaire peut être à la disposition des clients comme un site Web public, semi-public ou privé. ebXML Registry fournit ainsi un annuaire stable, où l'information publiée devient persistante"* (voir Figure 2.2 [3]).

Dans ce contexte, le contenu publié sous un annuaire ebXML inclut, entre autres : des schémas XML, des descriptions de processus, des descriptions de contexte, des modèles UML, des informations sur les entreprises et les composants logiciels, etc. La spécification du "*ebXML Registry Information Model*" (RIM) [80] définit les types de méta-données et des contenus qui peuvent être stockés dans un annuaire ebXML. Le document d'accompagnement "*ebXML Registry Services and Protocols*" (RS) définit les services fournis par un annuaire ebXML et les protocoles utilisés par les clients de l'annuaire pour interagir avec ces services.

3. www.javaworld.com/javaworld/jw-04-2005/jw-0425-webservices.html

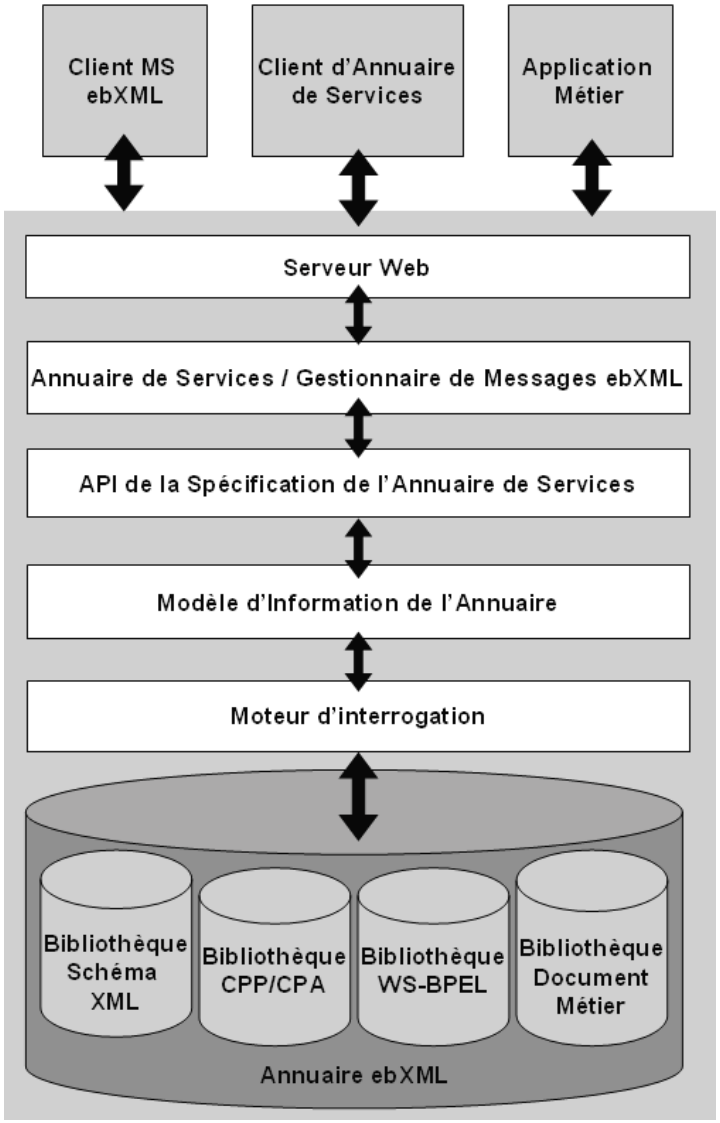


FIGURE 2.2 – Architecture basée sur un annuaire ebXML.

Selon les spécifications de RIM, un annuaire ebXML est capable de stocker tout type de contenu, tels que des documents XML, des documents textes, images, sons et vidéos. Les instances d'un tel contenu sont identifiées comme des *RepositoryItems* qui sont stockés dans un annuaire de contenu fourni par l'annuaire ebXML.

En plus de la *RepositoryItems*, un annuaire ebXML est également capable de stocker les méta-données normalisées qui peuvent être utilisées ultérieurement pour décrire les *RepositoryItems*. Les instances de ces méta-données sont identifiées comme des *RegistryObjects*, ou l'un de ses sous-types. Les *RegistryObjects* sont stockés dans l'annuaire prévu par l'annuaire ebXML.

2.3.2 SemRe : SemanticRegistry

SemRe [4] est une approche académique d'annuaire sémantique basé sur UDDI. Elle est développée dans le laboratoire Knoesis Center [5] de l'Université Wright State [6], SemRe est une plateforme qui permet essentiellement de stocker et de sélectionner des services en fonction de critères fonctionnels et non-fonctionnels.

L'annuaire prend en charge la sémantique des signatures des services Web. Il se base sur les annotations sémantiques qui assurent la description des relations explicites sémantiques entre les services. Les utilisateurs de l'annuaire peuvent créer des descriptions sémantiques pour les exigences et les capacités de chaque service Web. Pour ce faire, SemRe intègre les descriptions sémantiques des propriétés fonctionnelles des services en utilisant SAWSDL.

Pendant la phase de publication, la catégorisation des services est automatique. Les fournisseurs n'interviennent pas pour catégoriser les services Web qu'ils souhaitent publier.

La Figure 2.3 présente la correspondance entre les éléments descriptifs de SAWSDL et les éléments descriptifs au sein d'un annuaire UDDI. Le principe de cette approche est d'ajouter des informations sémantiques dans les structures de données UDDI. Ces informations sémantiques proviennent des annotations sémantiques sur les opérations. Un service Web SAWSDL correspond à une entité *Business Service* avec des références sur les opérations au niveau de la partie *Binding Template*. Chaque référence d'opération pointe sur un *Tmodel* dont la partie *CategoryBag* encapsule les informations sémantique de l'opération.

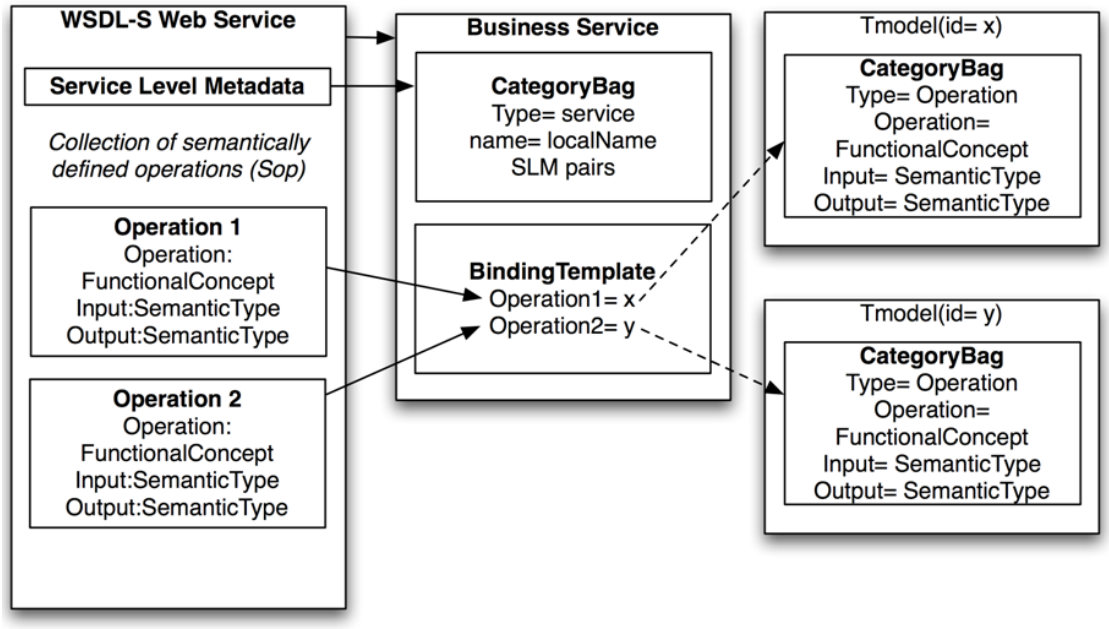


FIGURE 2.3 – La correspondance entre éléments SAWSDL et éléments UDDI.

Pendant la phase de découverte, un appariement est effectué entre une requête et les services publiés dans l'annuaire. Grâce aux descriptions des services et aux descriptions des critères requis, l'annuaire permet de sélectionner et de classer les services résultats selon certaines exigences requises. Les utilisateurs peuvent aussi préciser le degré d'appariement souhaité pour

4. SemRe : knoesis.wright.edu/opensource/ssr, visité en avril 2011
5. Knoesis Center : www.knoesis.org
6. Wright State University : www.wright.edu

chaque critère fonctionnel et/ou non-fonctionnel. Cet aspect a pour objectif de sélectionner des services même si un peu ou aucun service ne répond exactement à la description requise.

2.3.3 Dragon

Dragon est un annuaire distribué Open Source, il est basé sur UDDI et développé par la P.M.E. EBM WebSourcing [7] dans le cadre du consortium OW2 [8]. Le projet Dragon SOA fournit un ensemble complet de fonctionnalités qui visent la SOA à grande échelle.

Au moment de la conception des services Web, il offre un annuaire pour :
- stocker des informations sur les services, les contrats SLA et d'autres méta-données telles que les propriétés sémantiques,
- permettre une recherche de service et une découverte basées sur les méta-données, ainsi que la gestion du cycle de vie du service,
- fournir des fonctionnalités facilitant les mises à jour des processus et du service.

Au moment de l'exécution des services Web, il offre une interface de plateforme de services pour :
- appliquer les politiques des attributs de qualité de service (QoS),
- appliquer les SLA aux contrats des clients/fournisseurs,
- gérer les versions des services,
- suivre les indicateurs de SOA : QoS, la (ré)utilisation de service, le temps de développement, etc.

Pendant la phase de publication, la catégorisation des services est automatique. Les fournisseurs n'interviennent pas pour catégoriser les services Web qu'ils souhaitent publier.

Pendant la phase de découverte, un appariement est effectué entre des mots-clés précisés par l'utilisateur et des mots-clés associés aux services publiés dans l'annuaire.

2.3.4 FUSION Semantic Registry

FUSION Semantic Registry [51] est un annuaire de services Web basé sur UDDI. Le projet FUSION est un projet européen de recherche avec des partenaires à la fois des domaines académiques et industriels (SAP, Infomatix, Budapest University of Technology and Ecomonomics et d'autres [9]). Le projet a pour objectif de construire une solution basée sur la sémantique pour l'interopérabilité des applications métiers orientées services.

L'annuaire FUSION Semantic Registry est construit selon la spécification UDDI tout en l'enrichissant avec des informations de descriptions sémantiques. Pendant la phase de publication, la catégorisation des services est automatique. Les fournisseurs n'interviennent pas pour catégoriser les services Web qu'ils souhaitent publier. Il utilise ces mécanismes sémantiques afin d'améliorer les capacités de publication et de découverte. Pour ce faire, cet annuaire utilise la spécification SAWSDL afin de créer des descriptions annotées sémantiquement pour les interfaces de services. Il utilise aussi OWL-DL pour modéliser les capacités des services [49]. Et pour l'appariement de services, il se base sur un raisonnement des logiques de description (*DL Description Logics*) [51].

7. EBM WebSourcing : www.petalslink.com, visité en avril 2011
8. OW2 : www.ow2.org
9. Fusion Project : www.seerc.org/fusion/semanticregistry/fusion.html, visité en avril 2011

Pendant la phase de découverte, un appariement est effectué entre une requête et les services publiés dans l'annuaire. Pour réussir les appariements entre les services de l'annuaire, les auteurs de l'approche proposent d'utiliser une ontologie commune à tous les services. Elle sert comme un modèle sémantique partagé qui assure l'interopérabilité des services. La Figure 2.4 présente une table de correspondances entre les types des sémantiques utilisées en services, les éléments WSDL concernés, les attributs d'extensions sémantiques SAWSDL à utiliser et des concepts sémantiques proposées par l'ontologie de Fusion.

La Figure 2.4 précise, par exemple, que la sémantique fonctionnelle peut être spécifiée grâce à des annotations par des concepts de la taxonomie de classification au niveau de l'attribut *modelReference* dans l'élément WSDL *portType*. Quant à la sémantique comportementale, elle peut être spécifiée aux niveaux des éléments WSDL *portType*.

Type de la sémantique	L'ontologie FUSION	L'élément WSDL 1.1 à annoter	L'attribut d'extension SAWSDL
Sémantique Fonctionnelle	**Classification**	portType	modelReference
Sémantique de données	**Facette de données**	part	modelReference, loweringSchemaMapping liftingSchemaMapping
Sémantique Comportementale	**Ontologie d'état**	operation	modelReference

FIGURE 2.4 – Correspondances entre concepts de l'ontologie de Fusion et éléments WSDL [51].

Les auteurs fournissent une API associée à trois services Web :

1. *PublicationManager service* : un service Web pour la gestion de la publication des services,
2. *DiscoveryManager service* : un service Web pour la gestion de la découverte des services,
3. *AdminManager service* : un service Web pour l'administration de l'annuaire.

L'annuaire est implémenté sous forme d'application Web J2EE.

La Figure 2.5 présente l'architecture de FUSION Semantic Registry. Elle se compose de trois principales parties :

1. *Publication Manager Module* est un module dédié à la publication des services, il contient les APIs de la gestion de publication, la bibliothèque SAWSDL, la bibliothèque du client UDDI et la bibliothèque OWL.
2. *Discovery Manager Module* est un module dédié à la découverte des services, il contient les APIs de la gestion de découverte, la bibliothèque du client UDDI et la bibliothèque OWL.
3. *UDDI Server Module* est un module de serveur UDDI avec deux APIs de publication et d'interrogation pour communiquer avec les deux autres modules.

Les deux modules *Publication Manager Module* et *Discovery Manager Module* se partagent une base de connaissance OWL dite *OWL KB* et un *DL Reasoner*.

La publication d'une offre de service se compose de quatre étapes :

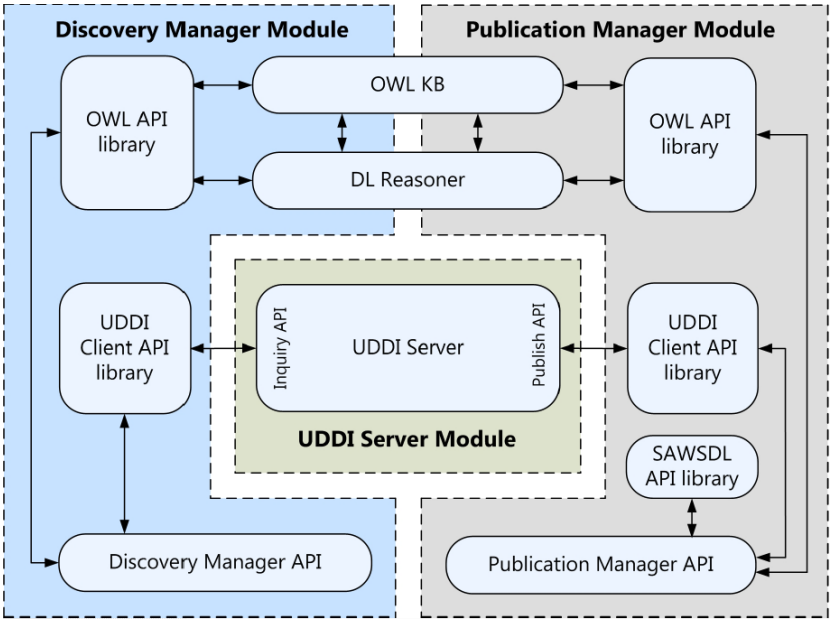


FIGURE 2.5 – Architecture de l'annuaire FUSION Semantic Registry[50].

1. Analyser (*Parser*) la description SAWSDL et extraire les informations syntaxiques et sémantiques,
2. Utiliser l'information sémantique extraite afin de construire un profile de service appelé *Advertisement Functional Profile (AFP)*.
3. Classifier l'AFP dans la base de connaissances *OWL KB* de l'annuaire,
4. Stocker (ou *Mapper*) l'information syntaxique et l'information sémantique correspondante vers les structures UDDI adéquates.

Une requête de découverte initialise un processus d'appariement sémantique et se compose de deux éléments :

1. Un URI faisant référence à un profil appelé *Request Functional Profile (RFP)*, il représente les caractéristiques du service Web requis,
2. En option, un identifiant ID faisant référence à un fournisseur de services en particulier qu'on souhaite contacter en premier,
3. Si le RFP est défini avec l'ontologie de FUSION, alors une requête syntaxique complémentaire de découverte *UDDIcompliant* est générée et soumise directement au serveur UDDI (qui passe par la *UDDI Client API library*).
4. Sinon si le RFP est défini avec une autre ontologie qui n'est pas partagée par le fournisseur, alors le module gestionnaire de découverte charge l'ontologie au *DL Reasoner* et traite la hiérarchie de relations sémantiques.

2.3.5 SWS ebXML

Shulte &al [82] propose une implémentation sémantique de freebXML [10] en se basant sur ebXML Registry Information Model (RIM) [80] (standardisé [11] par OASIS en ISO 15000).

10. freebXML : www.freebxml.org, visité en avril 2011
11. RIM : docs.oasis-open.org/regrep/v3.0/specs/regrep-rim-3.0-os.pdf

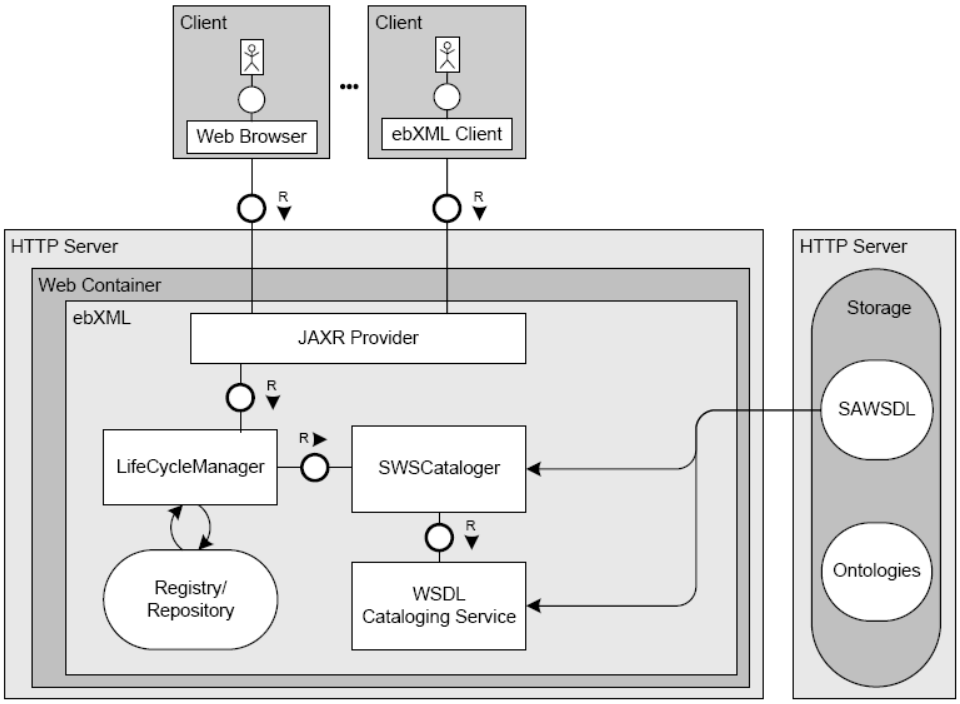


FIGURE 2.6 – Architecture ebXML étendue pour la publication [82].

La Figure 2.6 illustre les composants intervenant dans la phase de publication. Pour stocker les profiles des services Web, l'approche utilise certaines classes provenant du modèle d'information d'annuaire RIM. Un service Web est représenté par une instance de la classe *Service* qui contient une ou plusieurs instances de la classe *ServiceBinding*. Cette classe fournit des informations techniques telles que l'URI du service.

Afin d'intégrer les descriptions de services Web sémantiques (SWS) dans l'annuaire ebXML, les auteurs enrichissent le modèle d'information d'annuaire RIM par un nouvel élément de classification et son propre type, notés "*SWS*". Ce nouveau type permet de classifier les objets SWS, et parsuite distinguer entre les services Web sémantiques et les les services Web non-sémantiques. Les nouveaux types et composants qui les gèrent ont été implémentés en utilisant freebXML.

Pendant la phase de publication, la catégorisation des services est automatique. Les fournisseurs n'interviennent pas pour catégoriser les services Web qu'ils souhaitent publier. L'annuaire utilise les nouvelles classes de classification ainsi que leurs sous classes afin d'enregistrer la partie sémantique. Par ailleurs, les éléments WSDL correspondants sont publiés en utilisant les mécanismes de publication standard et habituels de ebXML. Pour ce faire, les auteurs proposent de chercher les descriptions SWS des services à partir des WSDL grounding comme celle fourni par OWL-S et WSMO.

Les descriptions de services SAWSDL peuvent être publiées sans aucune modification en utilisant le service standard de "catalogisation" de ebXML. Cependant, il est nécessaire d'enregistrer les informations sémantiques provenant de SAWSDL dans des éléments séparés. Ceci permet de différencier la découverte syntaxique de la découverte sémantique de services. Ces derniers aspects sont assurés par les composants *WSDL Cataloging Service* et *SWS Cataloger*. Toutes les données sont enregistrées dans le composant appelé *Registry/Repository*.

Concernant l'aspect "découverte", un annuaire ebXML habituel offre des capacités d'appariements syntaxiques basées sur des requêtes définies en utilisant SQL. La Figure 2.7 illustre les composants intervenant dans la phase de découverte.

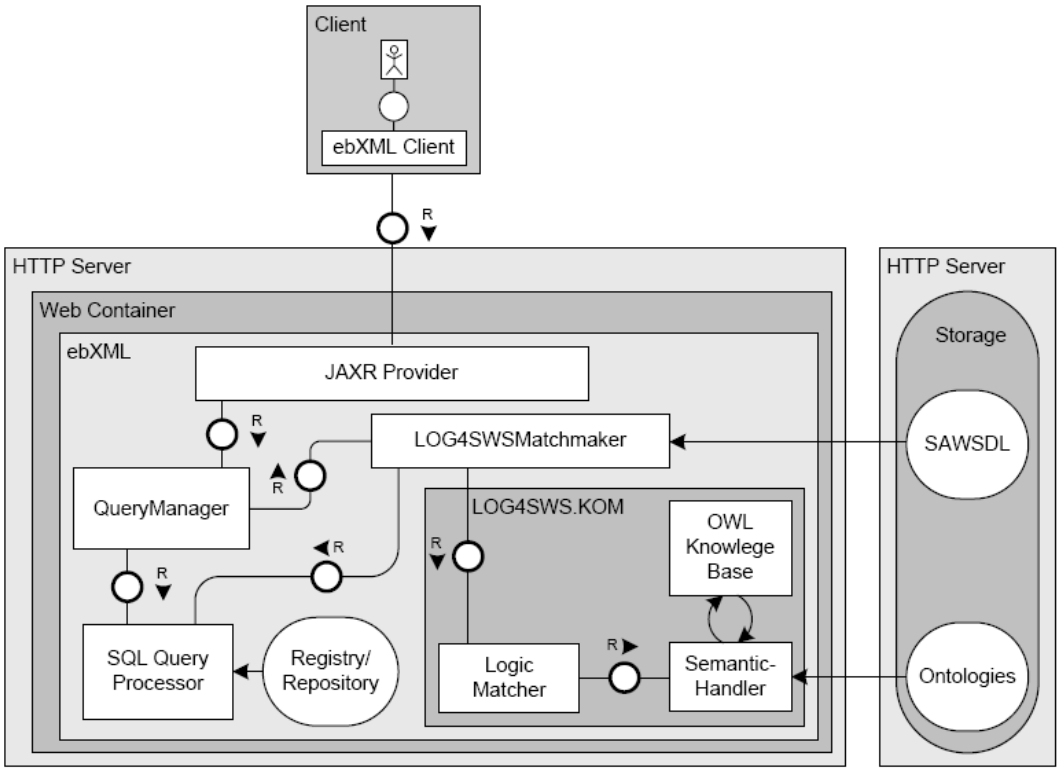


FIGURE 2.7 – Architecture ebXML étendue pour la découverte [82].

Pendant la phase de découverte, un appariement est effectué entre une requête et les services publiés dans l'annuaire. Et pour intégrer une découverte basée sur la sémantique, les auteurs ont ajouté de nouveaux mécanismes d'appariement sémantique appropriés. Au début du processus de la découverte, il y aura extraction de la sémantique SWS. Une requête SQL est créée par un utilisateur et envoyée au travers le client ebXML. Ensuite le matchmaker traite les descriptions SAWSDL des services en exécutant un algorithme d'appariement. En fin, le résultat est retourné au client ebXML. Le nouveau matchmaker ebXML est doté de deux méthodes : une première pour traiter les extensions de la requête SQL et une deuxième pour retourner une liste optionnelle d'informations additionnelles au client (telles que des valeurs de similarité de l'appariement).

2.4 Portails de services Web

Cette section donne un aperçu des portails publics qui se consacrent au stockage et à la découverte des services Web. Ces portails spécialisés recueillent des services Web publics soit en utilisant des moteurs d'exploration ciblée ou en se basant sur l'enregistrement manuel. Ils offrent essentiellement une fonctionnalité de recherche via une interface Web, certains offrent aussi une fonctionnalité de navigation. Dans la suite, je décris brièvement les principaux portails de services publics et qui s'appuient plutôt sur l'enregistrement manuel des services par les prestataires de services ou les utilisateurs du portail.

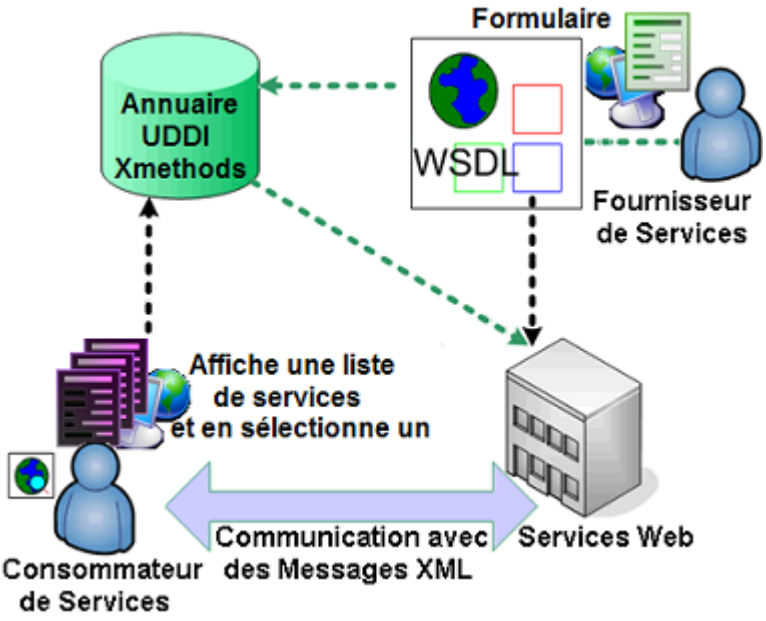


FIGURE 2.8 – Architecture basée sur l'annuaire Xmethods.

2.4.1 Xmethods

Xmethods [12] est probablement la plus ancienne référence d'accès public aux services Web. La publication et la découverte se font à travers des interfaces Web (voir Figure 2.8).

Pendant la phase de publication, la catégorisation des services est manuelle. Les fournisseurs interviennent pour catégoriser les services Web qu'ils souhaitent publier. D'un point de vue architecture, pour la publication, Xmethods utilise son propre annuaire UDDI. L'annuaire reçoit une requête HTTP de la part d'un fournisseur contenant les informations nécessaires pour un enregistrement basique d'un service Web selon l'approche UDDI. La structure UDDI est créée en à partir de données telles que le nom, la description WSDL et la localisation du service ainsi que des informations concernant le fournisseur.

D'un point de vue utilisation, uniquement des utilisateurs enregistrés sur le portail de l'annuaire peuvent publier leurs services Web en remplissant un formulaire. Le fournisseur du service Web doit préciser quelques informations telles que le nom du service, une description en ligne, l'URL du service, une adresse mail de contact, l'outil SOAP à utiliser, une description détaillée et des notes d'utilisations. Pendant la phase de découverte, un appariement est effectué entre des mots-clés précisés par l'utilisateur et des mots-clés associés aux services publiés dans l'annuaire. L'annuaire permet l'accès aux services à travers une interface UDDI. Il faut développer un client pour l'interface fournie. Ce client fait référence à l'URL : "*uddi.xmethods.net/inquire*" [79]. Le client peut être utilisé par l'outil en besoin des résultats de la découverte. D'un point de vue utilisation et en plus simple, pour trouver un service, on peut aussi consulter une page Web. La page interroge l'annuaire pour ensuite retourner simplement une longue liste des services Web disponibles. Pour chaque service Web, le portail peut afficher une page d'informations de base et quelques détails sur le service. La découverte se fait de façon manuelle par l'utilisateur en parcourant visuellement la page Web.

12. Xmethods : www.xmethods.com, visité en avril 2011

L'annuaire Xmethods permet de lister les services publiés. Les informations affichées sont les suivantes : le prestataire du service, le style (DOC, RPC, etc), le nom du service, une brève description textuelle, le type de son implémentation et un lien pour le tester. Pour aider l'utilisateur à mieux découvrir ce qu'il peut correspondre à sa demande, il peut accéder à la page d'informations détaillées, le contenu du document WSDL et une analyse de ce document.

D'après [60], un rapport fait à l'institut "Semantic Technology Institute" de l'Université de Innsbruck, seulement 82% des services Web publiés ont des documents WSDL valides. D'un point de vue qualitatif, le rapport expose deux exemples de requêtes par mot clé. Un premier exemple avec 92% de résultats potentiellement intéressants. Et un deuxième exemple avec uniquement 32% de résultats pertinents retournés par l'annuaire. De tels résultats donnent une idée sur les problèmes de précision de l'appariement entre les requêtes et les résultats, souvent dûs à l'aspect purement syntaxique de la découverte.

2.4.2 RemoteMethods

L'annuaire des services Web RemotheMethods [13] permet la publication, la classification et la découverte de services Web provenant de divers fournisseurs (entreprises ou particuliers). Il était déployé par *InfoGenius Inc.* depuis 1999 [61].

D'un point de vue architecture, pour la publication, RemotheMethods comme Xmethods utilise son propre annuaire UDDI. La structure UDDI est créée à partir de plusieurs de données toutes spécifiées par le fournisseurs à travers des formulaires en ligne. Pendant la phase de publication, la catégorisation des services est manuelle. Les fournisseurs interviennent pour catégoriser les services Web qu'ils souhaitent publier. L'annuaire RemotheMethods récupère des informations détaillées concernant l'entreprise ou le fournisseur, le service Web et le plan de tarification. Les développeurs du portail reçoivent les demandes de publication ensuite classifient les services Web par catégorie. La classification est à plusieurs niveaux mais elle reste assez vaste. L'annuaire RemoteMethods fournit un mécanisme permettant aux utilisateurs de poster des critiques et/ou évaluer les services Web [79].

Pendant la phase de découverte, un appariement est effectué entre des mots-clés précisés par l'utilisateur et des mots-clés associés aux services publiés dans l'annuaire. Les services peuvent être retrouvés en parcourant les catégories ou par une recherche syntaxique en utilisant des de mots-clés. Les résultats de la recherche peuvent être classés selon différents critères tels que le prix, la popularité, la date de mise à jour ou le classement par domaines ou par compagnie propriétaire du service. La description des résultats comporte le nom du service, son prix, un lien vers son WSDL, la date de sa publication, son score, des avis, un rapport d'erreur, une description textuelle et un lien de marquage dans une liste personnelle.

D'après [60], et d'un point de vue quantitatif, seulement 82% des services Web publiés ont des documents WSDL valides. D'un point de vue qualitatif, le rapport expose deux exemples de requêtes par mot-clé. Un premier exemple avec 75% de résultats potentiellement intéressants. Et un deuxième exemple avec 56% de résultats pertinents retournés par l'annuaire. L'annuaire RemoteMethods fait un peu mieux que l'annuaire Xmethods mais les mêmes problèmes persistent.

13. RemoteMethods : www.remotemethods.com, visité en avril 2011

2.4.3 StrikeIron

Depuis 2002, StrikeIron [14] assure la commercialisation de services Web et essaie de simplifier la publication, la découverte et la souscription aux services Web, autant du côté des prestataires de services, que du côté des utilisateurs [61]. StrikeIron utilise son propre annuaire UDDI afin de publier des services Web fournis par ses partenaires, principalement, Eloqua, IBM, Informatica, Microsoft et SalesForce. Pendant la phase de publication, la catégorisation des services est manuelle. Les fournisseurs interviennent pour catégoriser les services Web qu'ils souhaitent publier.

Pendant la phase de découverte, un appariement est effectué entre des mots-clés précisés par l'utilisateur et des mots-clés associés aux services publiés dans l'annuaire. Les services peuvent être retrouvés en parcourant des catégories de produits ou par une recherche syntaxique en utilisant des mots-clés. StrikeIron fournit des descriptions de services Web, plus compréhensibles que celles fournies par Xmethods et RemoteMethods. Les descriptions comprennent des listes de caractéristiques, de données et de documents associés aux services Web. L'interface Web permet de spécifier la demande sous forme de mots-clés, de découvrir les services les plus populaires ou de consulter les services et produits des partenaires. La description des résultats comporte un nom de service, des descriptions textuelles (fonctionnalités, prix et ressources techniques), un lien pour acheter le service et un lien pour le tester. Pour aider l'utilisateur à découvrir ce qu'il peut correspondre à sa demande, les résultats peuvent être ordonnés par nom ou organisés par domaine.

D'après [60], et d'un point de vue quantitatif, seulement 83% des services Web publiés ont des documents WSDL valides. D'un point de vue qualitatif, le rapport expose deux exemples de requêtes par mot-clé. Un premier exemple avec un seul pertinent résultat retourné par l'annuaire. Et un deuxième exemple avec uniquement 18% de résultats pertinents retournés par l'annuaire. L'annuaire fait pire que les annuaires RemoteMethods et Xmethods.

2.4.4 Seekda

Le portail de services Web Seekda [15] contient au moment de la rédaction du présent document (Avril 2011) 28.606 descriptions de services provenant de 7.739 fournisseurs.

Pendant la phase de publication, la catégorisation des services est manuelle. Les fournisseurs interviennent pour catégoriser les services Web qu'ils souhaitent publier. Seekda demande de fournir l'URL du WSDL du service ensuite il essaie d'extraire automatiquement des informations techniques et fonctionnelles à partir du WSDL. Ensuite, il propose au fournisseur d'ajouter des mots-clés (tags) et/ou une description compréhensibles par les utilisateurs potentiels du service.

Une fois le service Web enregistré, le fournisseur peut ajouter d'autres informations qui concernent son identité et/ou l'utilisation du service. Chaque service possède une fiche composée de quatre rubriques : *Overview* pour des informations générales telle que l'adresse URL, *Use Now* pour des détails sur les interfaces et les opérations du service, *Availability* pour des statistiques sur la disponibilité et *Comments* pour les commentaires des utilisateurs.

Le portail Seekda propose une recherche basique de services par mot-clé et une recherche avancée qui permet de découvrir les services à partir d'un nuage de tags, découvrir les fournisseurs par pays, découvrir les services Web les plus utilisés ou naviguer à travers les services

14. StrikeIron : www.strikeiron.com, visité en avril 2011
15. Seekda : webservices.seekda.com, visité en avril 2011

Web récemment découverts et sélectionnés. Les services peuvent être marqués pour une réutilisation/consultation ultérieure et ils peuvent être invoqués directement à partir du portail (pour tester directement le service web).

Pendant la phase de découverte, un appariement est effectué entre des mots-clés précisés par l'utilisateur et des mots-clés associés aux services publiés dans l'annuaire. Les services Web peuvent être découverts suite à un appariement syntaxique avec les mots-clés qu'on leurs a associés et/ou avec le pays ou le nom de leurs fournisseurs. Les résultats peuvent être ordonnés par pertinence, disponibilité, pays, fournisseur ou date de publication.

2.4.5 ProgrammableWeb

ProgrammableWeb [16] est un annuaire de services orienté communauté. La liste de services à découvrir ne contient pas que des descriptions de service en WSDL mais aussi des services RESTful, RSS et JSON. Pendant la phase de publication, la catégorisation des services est manuelle. Les fournisseurs interviennent pour catégoriser les services Web qu'ils souhaitent publier.

Pendant la phase de découverte, un appariement est effectué entre des mots-clés précisés par l'utilisateur et des mots-clés associés aux services publiés dans l'annuaire. En le comparant aux autres portails, la découverte de services dans ProgrammableWeb s'effectue en se basant sur une vue plus large de la description de services Web. ProgrammableWeb permet de spécifier la demande sous forme de mots-clés, ou de découvrir les mots -clés les plus utilisés (nuage de tags) et les services les plus populaires ou les plus récents. La description des résultats comporte un nom de service, une image, une description textuelle et un lien vers son API. Pour aider l'utilisateur à découvrir ce qu'il peut correspondre à sa demande, les résultats peuvent être ordonnés par nom, par date ou par popularité.

D'après [60], et d'un point de vue quantitatif 77% des services Web publiés ont des documents WSDL valides. D'un point de vue qualitatif, le rapport expose deux exemples de requêtes par mot-clé. Un premier exemple avec 100% de résultats retournés par l'annuaire. Et un deuxième avec uniquement 9% de résultats pertinents retournés par l'annuaire. L'annuaire RemoteMethods fait un peu mieux que l'annuaire Xmethods mais les mêmes problèmes persistent.

2.5 Analyse des approches de stockage

Ce chapitre a présenté l'état de l'art des annuaires de services.

Pour la synthèse, je dégage des critères d'évaluation de certains aspects des travaux présentés. Les critères sont en relation avec mes objectifs pour une contribution au stockage de services Web sémantiques. Dans la suite, j'examine les approches selon : la prise en charge de la sémantique et le degré d'automatisation (pendant les phases de publication et de découverte). La Table 2.1 présente les critères identifiés et les objectifs.

Pour la suite, la table 2.2 présente la sémantique de symboles utilisée pour évaluer les approches par rapport aux critères.

16. ProgrammableWeb : www.programmableweb.com

TABLE 2.1 – Objectifs d'une contribution en stockage de services Web sémantiques.

Critères	Objectifs
Sémantique	++ (Indépendance par rapport à la description syntaxique)
Automatisation	++ (Stockage et Découverte automatiques)

TABLE 2.2 – Sémantique des symboles utilisés dans l'examen des approches.

Symboles	-	+/-	+	++
Interprétations	Ne répond pas au critère	Il y a un manque	Répond mais on peut améliorer	Répond bien au critère

2.5.1 Prise en charge de la sémantique

Dans une architecture orientée service, un annuaire basé sur UDDI peut continuer à jouer le rôle d'un support de base. Mais pour assurer une découverte plus précise, il faut ajouter d'autres aspects fonctionnels, non-fonctionnels, sémantiques ou autres qui apportent a) un nouvel ensemble d'informations utiles et b) des mécanismes d'appariement et de sélection plus évolués que ceux offerts par UDDI. Les annuaires de services Web basés sur UDDI sont souvent présents dans l'offre standard des principaux fournisseurs SOA. Ils sont essentiels pendant les phases de conception et d'exécution en SOA. Malgré le succès de la spécification UDDI et son adoption rapide par l'industrie, les capacités de la découverte de services offerts sont assez limitées. Ceci est dû au manque de sémantique compréhensible par une machine qui permettra par la suite une découverte de services entièrement automatique et efficace.

Plusieurs travaux proposent d'étendre les annuaires UDDI avec des aspects non-fonctionnels comme la QoS, ou des aspects sémantiques. L'effort de l'approche de Dragon apporte plusieurs aspects fonctionnels et non-fonctionnels mais elle ne couvre pas la sémantique. L'effort de ebXML Registry a été approuvé par plusieurs industriels supportent ebXML Registry.

Les approches SemRe, FUSION et SWS ebXML ont pris en considération les données sémantiques et ont essayé d'ajouter les informations extraites des descriptions SAWSDL aux structures de données de leurs annuaires. Malgré les quelques réserves autour des cas d'ambiguïtés non résolus par SAWSDL que j'ai relevés durant mes travaux, ces approches proposent des idées à mieux élaborer et consolider pour passer des annuaires purement syntaxiques aux annuaires de services Web sémantiques tout en améliorant la publication et la découverte. Concernant l'aspect sémantique et de point de vue qualité, il est clair que des approches qui offrent des mécanismes d'analyse et/ou de stockage d'informations sémantiques offre une meilleur qualité de gestion de services Web sémantiques que les approches qui s'arrêtent à la gestion basée uniquement sur la syntaxe des descriptions et les mots-clés. De même, en termes de temps passé pour publier et/ou retrouver des services Web, les approches offrant des mécanismes dédiés pour traiter les informations sémantiques, elles permettent de réduire la durée des opérations de publication et de découverte. Ainsi, les approches SemRe, FUSION, Dragon et SWS ebXML qui sont dotées de mécanismes de publication et de découverte des services Web sémantiques, assurent un meilleur rapport qualité/temps que chez les autres approches purement syntaxiques. Ces dernières peuvent être améliorées si on reprend la description de service et on essaye de profiter d'avantage de ses capacités sémantiques pour faire évoluer la publication et la découverte.

En analysant les problèmes de précision relevés par des tests simples effectués sur les portails-annuaires de services Web dans [60], je conclue que la majorité, si ce n'est pas la totalité, des problèmes proviennent des mécanismes purement syntaxiques utilisés à la fois pendant la publication et la découverte. Ces problèmes confirment qu'il est difficile de s'attendre à ce que la sémantique d'un service Web se découvre par simple identification et/ou extraction de mots-clés à partir d'une définition syntaxique (WSDL ou autre) voir même d'une courte description humaine, ou grâce à un calcul de degré de pertinence basé sur des hyperliens ou des scores. Ces classements qui se veulent une aide à la découverte se basent sur des critères syntaxiques et numériques et ne tiennent absolument pas compte ni des structures des documents WSDL, ni de la disponibilité des services ou autres informations en relation avec les services découverts ou la requête. Un annuaire de services Web de nos jours doit être capable de réaliser la découverte tout en prenant en charge la sémantique du service surtout au niveau fonctionnel (ex. opérations, inputs et outputs).

La Table 2.3 résume l'examen des approches de stockage de services Web selon ce critère.

TABLE 2.3 – Prise en charge de la sémantique par les annuaires de services.

Critères	Objectifs	Dragon	SemRe	FUSION	SWS ebXML	Portails
Sémantique	++	-	+	+	+	-

2.5.2 Degré d'automatisation

En ce qui concerne la découverte de services sur les portails-annuaires, je peux dire que les résultats ne sont pas très prometteurs pour une découverte pertinente et dynamique des services Web. Ceci est dû en grande partie aux diverses manipulations et vérifications manuelles nécessaires avant qu'un utilisateur puisse juger que tel ou tel service correspondrait à sa demande. En plus, les portails n'offrent pas une découverte assez rapide et efficace pour des environnements dynamiques et à grande échelle. Souvent, les fournisseurs et les utilisateurs effectuent des interactions manuelles afin d'utiliser les portails-annuaires de services.

Il est vrai que les portails de services offrent aux utilisateurs une large couverture, mais ceci les mène souvent - les utilisateurs - à passer un temps considérable en navigant dans les résultats. Ainsi vu, le filtrage des services Web ne peut donc se faire d'une façon efficace. Néanmoins, en raison de leur spécialisation, les portails sont un moyen facile et ergonomique pour trouver des services seulement si le temps et la manipulation manuelle ne représentent pas d'inconvénients pour la découverte, mais ceci n'est pas le cas des applications dynamiques et à grande échelle, comme dans les applications industrielles ou de recherche. Tous les portails de services Web précédemment évoqués nécessitent que les prestataires de services prennent le temps pour manipuler et décrire leurs services, d'une part, et que les utilisateurs mènent manuellement le processus de découverte, d'autre part.

Les approches SemRe, FUSION, Dragon et SWS ebXML sont dotées de plus de mécanismes pour pouvoir automatiser la publication et la découverte des services Web sémantiques, mais chacune continue à dépendre parfois de l'intervention de l'utilisateur pour valider ou débloquer que ce soit les processus et les résultats (comme dans SWS ebXML et Dragon pour choisir/sélectionner un service) ou faire des choix intermédiaires sur les degrés d'appariements (comme dans SemRe).

Je reprend ici les résultats et les problèmes évoqués pour les approches présentés à la fin des sous sections 2.4.5, 2.4.3, 2.4.2 et 2.4.1. En analysant ces problèmes de précision relevés par des tests simples effectués sur les portails-annuaires de services Web dans [60], je conclue qu'il est difficile de s'attendre à ce que les consommateurs de services puissent de manière pertinente découvrir des services en parcourant visuellement une liste de résultats potentiellement intéressants et qu'à la fin et après beaucoup de temps de vérification, uniquement 2 ou 3 services peuvent être adéquats à sa demande.

La Table 2.4 résume l'examen des approches de stockage de services Web selon ce critère.

TABLE 2.4 – Degré d'automatisation des annuaires de services.

Critères	Objectifs	Dragon	SemRe	FUSION	SWS ebXML	Portails
Automatisation	++	+/-	+/-	+	+/-	-

2.5.3 Récapitulatif

Les efforts existants basés sur l'approche des annuaires sont principalement utilisés avec les descriptions de service WSDL. De telles approches ont trouvé un succès dans des environnements industriels fermés, mais elles n'ont pas réussi à s'imposer avec ampleur dans les environnements publics ouverts. En fait, ces environnements sont souvent dynamiques et pas assez adaptés à une découverte plus ou moins intuitive.

La Table 2.5 résume l'examen des approches de stockage de services Web selon les différents critères. Selon cette table, aucune des approches examinées ne réalise mes objectifs. Il est clair que l'aspect syntaxique d'une description peut être toujours traité automatiquement et de continuer à améliorer les mécanismes dédiés à cet aspect. Mais, il faut absolument proposer des nouveaux composants dans l'architecture d'un annuaire de services Web afin qu'il puisse traiter convenablement la description sémantique qui ne peut être traitée en aucun cas comme l'aspect syntaxique vu la nature elle-même de cet aspect de la description.

TABLE 2.5 – Table récapitulative de l'examen des approches de stockage de services Web.

Critères	Objectifs	Dragon	SemRe	FUSION	SWS ebXML	Portails
Sémantique	++	-	+	+	+	-
Automatisation	++	+/-	+/-	+	+/-	-

En conclusion, je pense qu'un travail sur une solution pour le stockage et la découverte de services peut exploiter tout ce qui était acquis par les efforts d'extension des annuaires, sous condition qu'elle présente une alternative innovante au niveau du mécanisme d'appariement et de sélection. Une telle solution doit faire mieux que les annuaires traditionnels et les portails-annuaires Web ; car de nos jours, les services Web laissent place aux services Web sémantiques. Ceci offre plus d'idées mais exige aussi plus d'efficacité de la part des algorithmes et des modules des annuaires de services.

2.6 Conclusion

J'ai présenté dans ce chapitre l'état de l'art des annuaires de services. J'ai commencé par introduire les concepts d'annuaire et de portail-annuaire de services Web. Ensuite, j'ai présenté les principaux travaux d'annuaires basés sur des standards ainsi que les principaux portails de services Web. Enfin, j'ai examiné les approches par rapport au critère de la prise en compte de la sémantique et au critère d'automatisation, pendant les phases de publication et de découverte. Dans le chapitre suivant, je présente l'état de l'art de la découverte de services Web.

Chapitre 3

Découverte de Services

Sommaire

3.1 Introduction

La découverte de services publiés dans un annuaire se base sur un besoin présenté sous forme de requête. Le principe de la découverte est simple : identifier les services qui peuvent répondre à la requête. Pour cela, une façon de faire serait d'identifier un degré de similitude entre les différents concepts sémantiques qui décrivent le service requis (la requête) et celles des services offerts (les services publiés). Par exemple pour *WSDL*, tous les principaux éléments des services

doivent être pris en considération dans cette procédure, à savoir : *interface, operation, input* et *output*. Un tel mécanisme d'évaluation pour la découverte est appelé : *Matching* (Appariement).

Dans la littérature, et comme mentionné dans [83], il existe trois catégories d'approches de découverte de services Web : les approches logiques, les approches non-logiques et les approches hybrides. Les approches de découverte qualifiées de non-logiques sont basées sur le calcul du degré de similarité textuelle à partir de graphes structurés construits à cet effet ou encore le calcul de distance (le chemin) entre concepts. Tandis que les approches de découverte logique sont basées essentiellement sur des approches déductives, les approches qui combinent les mécanismes logiques et non-logiques sont qualifiées d'hybrides. Nous n'examinons pas dans ce chapitre les approches de découverte selon cette classification. Nous identifions un peu plus loin dans ce chapitre nos propres critères d'examen et de comparaison des approches de découverte dans cet état de l'art. Nous n'évoquons cette classification que pour la présentation des approches afin que le lecteur puisse retrouver des repères clairs tout au long de la première partie du chapitre.

Après avoir présenté, dans le chapitre 1 les approches et les modèles de description, puis les travaux autour du stockage des descriptions de services Web dans le chapitre 2, je présente dans ce chapitre les différentes approches et modèles ainsi que les plateformes les plus connues de découverte de services que je considère dans mes travaux. Je commence par exposer les approches de découverte non-logiques (Section 3.2), puis les approches de découverte logiques (Section 3.3) et finalement les approches de découverte hybrides (Section 3.4). Une synthèse est présentée à la fin du chapitre (Section 3.5), dans laquelle j'identifie notamment les limitations de certaines approches vis-à-vis du problème de la découverte de services sémantiques.

3.2 Les approches non logiques

3.2.1 Principe

Certaines approches de découverte se fondent sur un appariement non logique. Ce dernier utilise des mécanismes syntaxiques, structurels et numériques tels que le concept de distance numérique, l'appariement de graphes structurés, la similitude syntaxique, etc. L'idée principale est d'exploiter la sémantique implicite plutôt que la sémantique explicite. Pour assurer cela, de tels mécanismes d'appariement utilisent les fréquences des termes, les sous-graphes, etc.

Dans la suite de cette section, je présente, comme exemples de systèmes basés sur cette catégorie de découverte, les travaux autour de iMatcher1 [83] et DSD-Matchmaker [42] de la description de services DIANE (DSD) [53].

3.2.2 iMatcher1

iMatcher1 [83] offre un système de découverte non logique grâce à un *matchmaker* (apparieur) syntaxique basé sur les profils de services. Il prend en entrée un ensemble de profils de service spécifiés en OWL-S qui sont stockés sous forme de graphes RDF sérialisés dans une base de données RDF avec une extension de RDQL, appelée iRDQL [8]. Le degré d'appariement est calculé à partir de quatre métriques de similarité syntaxique de recherche d'information : TFIDF (Term Frequency-Inverse Document Frequency) [34, 81], la distance de similarité

de Levenshtein [62, 29, 63], la mesure du vecteur Cosinus [27, 95], la mesure de divergence de Jensen-Shannon [26]. Les résultats sont classés en fonction des scores numériques de ces mesures de similarités syntaxiques et d'un seuil défini par l'utilisateur. Dans la sous-section 3.4.3, je présente aussi iMatcher2 le successeur hybride de iMatcher1.

3.2.3 DSD-matchmaker

DSD-matchmaker [42] permet une découverte basée sur l'appariement de graphes entre deux descriptions de service spécifiées dans le langage de description de service orienté objet DSD [53] qui spécifie des variables et des ensembles d'objets déclaratifs sans aucune sémantique basée sur la logique. La description est composée de deux parties : une partie statique déclarée dès le début et une partie dynamique, construite à base d'informations liées au contexte du service. Le processus d'appariement détermine les variables à satisfaire, découvre en se basant sur l'état des services, celui qui correspond le mieux parmi l'ensemble de services acceptés par la requête, et renvoie une valeur numérique représentant le degré d'appariement de service DSD.

Le DSD-matchmaker procède en plusieurs étapes dans lesquelles il effectue des estimations sur les offres. D'abord, il détermine les valeurs concrètes des entrées (*inputs*) à utiliser pour l'exécution d'estimation. Cela permet également de recueillir des informations pour savoir si une estimation de l'offre en cours est prometteuse, c'est-à-dire si la valeur fournie par cette opération est utile pour décider à quel niveau cette offre correspond à une requête donnée. Dans une deuxième étape, le DSD-matchmaker exécute seulement les estimations prometteuses et met à jour les descriptions des services offerts à l'aide d'informations collectées dynamiquement. Finalement, dans une troisième étape, l'appariement final est effectué sur la base des descriptions mises à jour. Dans [54], les auteurs expliquent leur approche pour intégrer des informations dynamiques dans les descriptions des services.

D'après [56], le DSD-matchmaker parcourt en parallèle deux descriptions (celle de l'offre en cours et celle de la requête) sous forme d'arbres et compare récursivement les nœuds de ces deux graphes. L'algorithme d'appariement de graphe utilisé pour calculer les degrés d'appariement est donné dans [42, 43, 44]. Le degré d'appariement de deux nœuds est une agrégation du degré d'appariement des deux types des concepts sémantiques représentés par les deux nœuds et du degré d'appariement des propriétés de ces deux concepts.

3.2.4 Discussion

La découverte basée sur l'appariement non logique, trouve ses origines dans les mécanismes de recherche d'information. L'inconvénient majeur des approches non logiques est qu'elles relèvent toujours des problèmes linguistiques, terminologiques ou statistiques. Elles se basent souvent sur des hypothèses et leurs limites refont surface dès que les descriptions de services ne respectent pas les hypothèses posées. Des résultats de découverte non pertinents peuvent être causés par plusieurs problématiques bien connues dans ce domaine, telles que les mauvaises et les fausses interprétations de la syntaxe. Ceci peut être dû par exemple à une morphologie inadaptée des termes, la richesse sémantique d'une langue, etc.

Pour faire face aux limites de l'appariement basé sur des approches non logique (telles que la linguistique, la terminologie et la statistique), certaines approches essaient de profiter du progrès

technologique (capacité de stockage et rapidité de traitement). Elles accroissent tant que possible les performances de leurs algorithmes et leurs techniques, mais ceux-ci restent insensibles aux problématiques précédemment évoquées, et face auxquelles il manque ainsi des mécanismes de découverte plus adaptés. Mais face à l'évolution des moyens d'expression de la sémantique des services, certaines approches choisissent souvent de doper leurs techniques par quelques mécanismes logiques d'appariement et elles finissent, ainsi, par proposer des solutions hybrides offrant des meilleurs résultats dans les cas où la syntaxe ne suffit pas.

3.3 Les approches logiques

3.3.1 Principe

Après avoir présenté, dans les sections précédentes, des approches de découverte de services basées sur les mécanismes non logiques, je me focalise à partir de maintenant sur les approches de découverte basées sur la logique. Quand je parle d'approches de découverte basées la logique, je m'intéresse à celles qui utilisent des descriptions de services et des requêtes spécifiées en langages basés sur des formalismes logiques, telles que la logique de description, la logique de premier ordre, etc.

Cette catégorie utilise des concepts d'ontologies et des règles logiques. Les degrés d'appariement sont déterminés de différentes façons et en fonction de la sémantique des éléments des descriptions à apparier. Il existe principalement trois approches d'appariement :

- IO-matching : dit aussi "IO-matching de profil de service". Ce type d'appariement est déterminé à partir des données sémantiques des paramètres de service : les entrées : *inputs* (I) et les sorties : *outputs* (O). Ce type de d'appariement est adopté dans [87], [23] et [78].
- PE-matching : Ce type d'appariement est déterminé à partir d'appariements sur des préconditions (P) et des effets (E) des services et des requêtes. Ce type d'appariement est adopté dans PCEM [83] avec des préconditions et des effets spécifiées en Prolog.
- IOPE-matching : Ce type d'appariement est déterminé à partir d'appariements sur les données sémantiques des inputs (I), des outputs (O), des préconditions (P) et des effets (E) des services et des requêtes. Cet appariement est adopté dans [32], [37], [55] et [88].

Dans cette section, je présente des approches logiques d'appariement pour la découverte ([78], [32], [37] et [55]). Certains de ces travaux sont basés sur des langages de description de service déjà présentés dans le chapitre présentant l'état de l'art des descriptions de services (Chapitre 1).

3.3.2 WSC : une approche basée sur DAML

Dans [78], l'appariement sémantique de WSC (Web Services Capabilities) est basé sur des ontologies DAML [18]. Les services publiés et les requêtes se réfèrent à des concepts DAML et une sémantique associée. Un service publié correspond à une requête lorsque tous les outputs de la requête correspondent à des outputs du service publié, et tous les inputs du service publié correspondent à des inputs de la requête.

Partant d'un appariement basé sur DAML, l'approche se base sur un IO-matching, c'est-à-dire, un processus d'appariement qui ne considère que les inputs et les outputs. Les auteurs de [78] proposent quatre degrés d'appariement : EXACT, PLUGIN, SUBSUMES et FAIL. La Figure 3.1

présente l'algorithme des règles d'affectation des degrés d'appariement des outputs. Soient A un service publié, R une requête de service, et outA et outR deux de leurs outputs respectives. L'algorithme précise que, pour apparier des outputs de A et de R, le degré EXACT est défini lorsque les deux outputs à apparier sont équivalents ou lorsque un output de R est annoté par un sous-concept de celui qui annote un output de A (lignes 2 et 3). Le degré PLUGIN est identifié si un output de A est plus générique qu'un output de R (ligne 4). Le degré SUBSUMES est identifié si un output de R est plus générique qu'un output de A (ligne 5). Le degré FAIL est identifié quand aucune relation hiérarchique entre les deux outputs à apparier n'est identifiée (ligne 6).

```
1 degreeOfMatch(outR,outA):
2    if outA=outR then return exact
3    if outR subclassOf outA then return exact
4    if outA subsumes outR then return plugIn
5    if outR subsumes outA then return subsumes
6    otherwise fail
```

FIGURE 3.1 – Règles d'affectation des degrés d'appariement d'outputs dans WSC [78].

Le principal critère de tri dans WSC est la sélection du meilleur score au niveau des outputs. L'appariement des inputs n'est utilisé qu'en deuxième lieu au cas où il y aurait une égalité sur les appariements des outputs. L'algorithme d'appariement compare la demande aux services publiés. Il apparie un à un leurs outputs et leurs inputs et enregistre ceux qui correspondent le plus.

La Figure 3.2 présente l'algorithme des règles de tri des scores dans WSC [78]. Soient deux services offerts S1 et S2. Les scores match1 et match2 sont les résultats de l'appariement de chaque service avec une requête donnée. L'algorithme trie ici ces deux scores d'appariement (ligne 1). Si le degré appariement d'output du premier score est plus grand que le degré d'appariement d'output du deuxième score alors le premier score est plus grand que le deuxième (ligne 2). Si le degré appariement d'output du premier score est égal au degré d'appariement d'output du deuxième score et si le degré appariement d'input du premier score est plus grand que le degré d'appariement d'input du deuxième score alors le premier score est plus grand que le deuxième (ligne 3 et 4). Si le degré appariement d'output du premier score est égal au degré d'appariement d'output du deuxième score et si le degré appariement d'input du premier score est égal au degré d'appariement d'input du deuxième score alors le premier score est égal au deuxième (ligne 5 et 6).

```
1 sortRule(match1,match2) {
2    if match1.output > match2.output then match1 > match2
3    if match1.output = match2.output
4       & match1.input > match2.input then match1 > match2
5    if match1.output = match2.output
6       & match1.input = match2.input then match1 = match2
```

FIGURE 3.2 – Règles de tri des scores dans WSC.

3.3.3 OWLS-M : OWL-S Matchmaker

Dans [32], les auteurs proposent une approche basée sur le IOPE-matching. Partant d'un appariement entre deux profils de service spécifiés en OWL-S, le processus d'appariement prend en compte les inputs, les outputs, la catégorie de service et des éléments pré-définis par l'utilisateur.

La Figure 3.3 résume le processus de l'agrégation. Elle détaille le processus d'appariement qui comporte quatre tâches : l'appariement des inputs (Figure 3.3.(1)), l'appariement des outputs (Figure 3.3.(2)) et l'appariement de la catégorie du service (Figure 3.3.(3)). L'algorithme calcule dans chaque tâche le degré d'appariement. Une quatrième tâche pendant laquelle des contraintes et des fonctionnalités pré-définies par l'utilisateur s'appliquent (Figure 3.3.(4)) avant que les résultats soient agrégés afin de retourner le résultat final (Figure 3.3.(5)). Le résultat est soit un échec s'il y a échec d'appariement dans l'une des précédentes tâches (Figure 3.3.(6)), soit une valeur numérique (dite *RANK*) déterminée par l'agrégation des degrés d'appariements de chaque tâche (Figure 3.3.(7)) et représentant le rang du service Web apparié dans l'ensemble des services offerts.

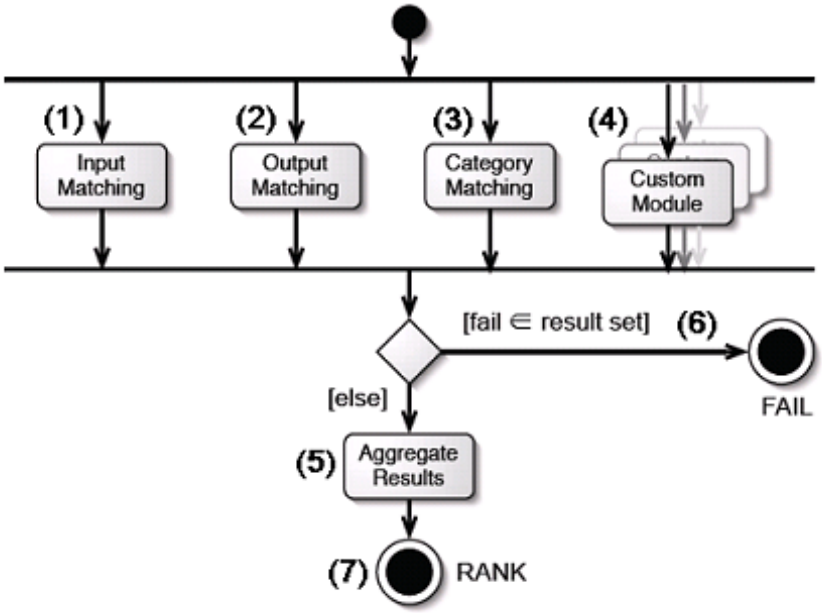


FIGURE 3.3 – Agrégation des degrés d'appariement dans OWLS-M [32].

Pour évaluer l'appariement, les auteurs proposent quatre types de degrés d'appariement avec quelques différences d'interprétation d'un élément du profil à un autre :

- Degrés d'appariement pour les input / outputs :
  - 0 : FAIL : si au moins un input / output requis n'a pas été apparié,
  - 1 : UNKNOWN : si l'algorithme d'appariement ne peut pas catégoriser un input / output,
  - 2 : SUBSUMES : si l'input / l'output offert est plus spécifique que l'input / l'output requis,
  - 3 : EQUIVALENT : en cas d'équivalence entre inputs / outputs.

.

- Degrés d'appariement pour les catégories de service :
  - 0 : FAIL : si les concepts des catégories de service ne s'apparient pas,
  - 1 : UNKNOWN : si l'algorithme d'appariement ne peut pas catégoriser l'un des services,
  - 2 : SUBSUMES : si le service offert est plus spécifique que le service requis,
  - 3 : EQUIVALENT : en cas d'équivalence entre catégories de service.

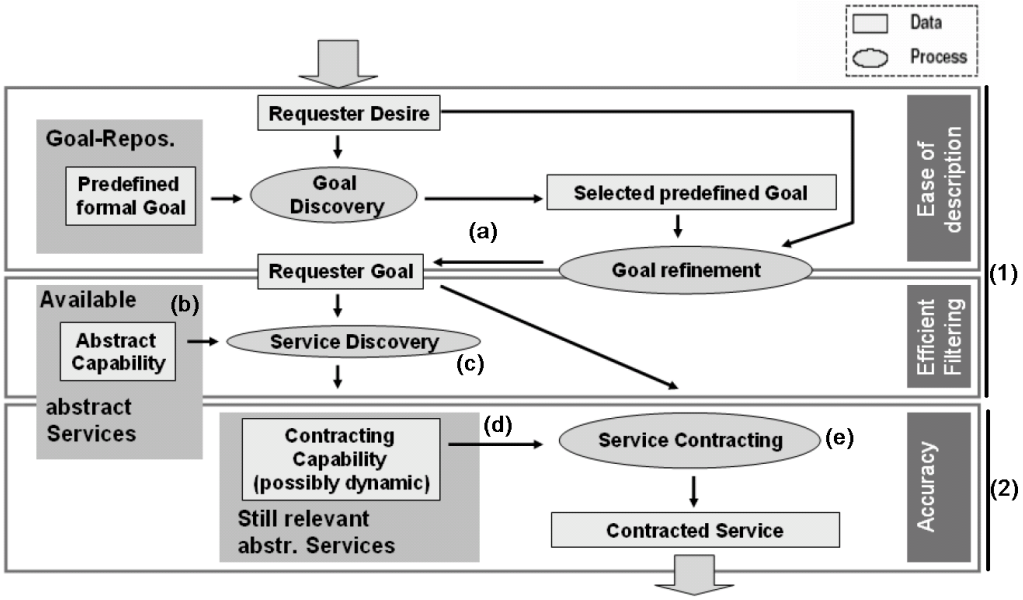


FIGURE 3.4 – Un modèle conceptuel pour la procédure de découverte [37].

3.3.4 ALS : Automatic Location of Services

Dans [37], les auteurs proposent un modèle conceptuel (Figure 3.4) pour une découverte sémantique (dite localisation sémantique) de services qui permet de réutiliser des objectifs (Goals) prédéfinis, des abstractions de services et finalement des services concrets pour répondre à l'objectif d'une requête.

Cela s'effectue en deux grandes phases :

- La phase de découverte de services (Figure 3.4.(1)) s'intéresse à :
  - des objectifs représentant ce que l'utilisateur souhaite réaliser après utilisation du service, et ce qu'il recevrait comme output et effets du service(Figure 3.4.(a)),
  - des descriptions abstraites plutôt qu'à des descriptions précises de capacités (Figure 3.4.(b)),
  - l'identification des services candidats possibles (Figure 3.4.(c)).
- La phase de contractualisation de service (Figure 3.4.(2)) s'intéresse aux :
  - descriptions précises des capacités (uniquement celles des services découverts dans la première phase) (Figure 3.4.(d)),
  - informations sur les inputs, pour avoir un service concret (avec tous les inputs, les préconditions et les postconditions) (Figure 3.4.(e)),
  - interactions entre le fournisseur et le demandeur de service.

Intention of $\mathcal{G}$ / $\mathcal{A}$	$I_{\mathcal{A}} = \forall$		$I_{\mathcal{A}} = \exists$	
$I_{\mathcal{G}} = \forall$	$R_{\mathcal{G}} = R_{\mathcal{A}}$	**Match**	$R_{\mathcal{G}} = R_{\mathcal{A}}$	**PossMatch**
	$R_{\mathcal{G}} \subseteq R_{\mathcal{A}}$	**Match**	$R_{\mathcal{G}} \subseteq R_{\mathcal{A}}$	**PossMatch**
	$R_{\mathcal{G}} \supseteq R_{\mathcal{A}}$	**ParMatch**	$R_{\mathcal{G}} \supseteq R_{\mathcal{A}}$	**ParMatch**
	$R_{\mathcal{G}} \cap R_{\mathcal{A}} \neq \emptyset$	**ParMatch**	$R_{\mathcal{G}} \cap R_{\mathcal{A}} \neq \emptyset$	**PossParMatch**
	$R_{\mathcal{G}} \cap R_{\mathcal{A}} = \emptyset$	**Nonmatch**	$R_{\mathcal{G}} \cap R_{\mathcal{A}} = \emptyset$	**Nonmatch**
$I_{\mathcal{G}} = \exists$	$R_{\mathcal{G}} = R_{\mathcal{A}}$	**Match**	$R_{\mathcal{G}} = R_{\mathcal{A}}$	**Match**
	$R_{\mathcal{G}} \subseteq R_{\mathcal{A}}$	**Match**	$R_{\mathcal{G}} \subseteq R_{\mathcal{A}}$	**PossMatch**
	$R_{\mathcal{G}} \supseteq R_{\mathcal{A}}$	**Match**	$R_{\mathcal{G}} \supseteq R_{\mathcal{A}}$	**Match**
	$R_{\mathcal{G}} \cap R_{\mathcal{A}} \neq \emptyset$	**Match**	$R_{\mathcal{G}} \cap R_{\mathcal{A}} \neq \emptyset$	**PossMatch**
	$R_{\mathcal{G}} \cap R_{\mathcal{A}} = \emptyset$	**Nonmatch**	$R_{\mathcal{G}} \cap R_{\mathcal{A}} = \emptyset$	**Nonmatch**

FIGURE 3.5 – Les degrés d'appariement intentionnel [37].

Cette approche propose six degrés d'appariement appelés "degrés d'appariement intentionnel" (voir la Figure 3.5). Ils sont définis particulièrement comme suit :

- Match : Le service offert satisfait complètement la requête,
- PossMatch : Le service offert peut satisfaire complètement la requête, mais à cause d'une incomplétude dans les descriptions, ceci ne peut être garanti si on se limite aux informations disponibles. Ce degré d'appariement est identifié à la phase de contractualisation,
- ParMatch : Le service offert satisfait la requête, mais partiellement. Il offre certains aspects requis mais pas la totalité.
- PossParMatch : Le service offert peut satisfaire partiellement la requête, mais à cause d'une incomplétude dans les descriptions, ceci ne peut être garanti si on se limite aux informations disponibles. Ce degré d'appariement est identifié à la phase de contractualisation,
- NoMatch : Aucun des précédents cas n'est identifié, l'offre est complètement non pertinente par rapport à la requête.

3.3.5 L'approche de la pertinence graduée (GR : Graded Relevance)

Dans [55], les auteurs ont travaillé sur l'approche présentée dans [37], ils proposent des extensions au niveau des degrés d'appariement. Ils ajoutent :

- RelationMatch : Le service offert ne fournit pas les services requis mais plutôt des fonctionnalités en relation avec eux. Ceci peut être utile si une coordination peut être planifiée avec d'autres services offerts.
- ExcessMatch : Le service offert est capable de fournir les services requis mais des effets additionnels indésirables et non requis par l'utilisateur se produiront.

3.3.6 Discussion

Je discute ici de quelques points soulignés dans les approches logiques citées dans cette section. Dans l'approche de WSC (Section 3.3.2), j'ai souligné deux éléments de discussion que je reprends ici :

- Selon la sélection de service proposée par les auteurs, l'appariement ne serait pas suffisant pour juger que tel ou tel service est pertinent pour une requête, vu qu'à la fin ce n'est que la valeur d'appariement des outputs qui compte. Que ce soit les outputs seuls ou avec les

inputs, ces paramètres fonctionnels ne présentent à eux seuls qu'une partie d'une solution pour la découverte. D'autres éléments, comme la catégorie de service par exemple, peuvent jouer un rôle essentiel dans l'appariement.
– Il n'est pas suffisant pour assurer un processus d'appariement efficace de permettre aux demandeurs de services de fixer le degré minimal (seuil) d'appariement ou de restreindre les concepts dans lesquels la recherche serait effectuée. Cela conduit souvent à un échec d'appariement à cause d'un mauvais choix du seuil ou des concepts. Par la suite, le demandeur de service serait amené à exécuter à nouveau une ou plusieurs fois la requête.

Dans l'approche OWLS-M (Section 3.3.3), j'ai souligné trois éléments de discussion que je reprends ici :

– Cette approche n'exprime pas clairement sa solution au problème d'ambigüité de la relation d'inclusion représentée par le degré d'appariement SUBSUMES. L'interprétation de ce degré ne spécifie pas si l'inclusion entre deux concepts est uniquement directe ou peut être aussi indirecte ? Il est intéressant de le savoir car :
  – si c'est le premier cas, alors une inclusion indirecte, considérée souvent par les autres approches comme un cas particulier d'appariement, est évaluée par OWLS-M comme étant un échec d'appariement.
  – si c'est le deuxième cas, la définition du degré SUBSUMES serait floue car elle couvrirait en même temps un appariement imprécis (l'inclusion directe) et un appariement encore moins précis (l'inclusion indirecte).
– Un deuxième point concerne l'inclusion inverse. Elle est vue par OWLS-M comme un cas d'échec d'appariement alors que d'autres approches la prennent en compte. L'inclusion inverse est identifiée quand (1) les inputs requis sont plus génériques que les inputs offerts ou quand (2) les outputs offerts sont plus génériques que les outputs requis.
– Le troisième point concerne l'intervention de l'utilisateur dans le processus d'appariement qui n'est pas toujours possible dans les environnements de découverte automatique et dynamique de services.

3.4 Les approches hybrides

3.4.1 Principe

Les approches hybrides utilisent une combinaison de mécanismes logiques et non logiques. L'idée est de remédier à certaines limites de chacun de ces deux mécanismes grâce à différentes combinaisons hybrides qui réussissent là où chacun de ces deux mécanismes échoue.

3.4.2 OWLS-MX

OWLS-MX [45] est un matchmaker sémantique hybride qui exploite le raisonnement logique et les techniques de recherche d'information pour réaliser un IO-matching entre des profils de services OWL-S.

En plus des degrés de succès et d'échec d'appariement (Exact et Fail), OWLS-MX propose cinq degrés d'appariement : les trois premiers sont uniquement basés sur la logique (Plug-In,

Subsumes et Subsumed-By) et les deux derniers sont hybrides. Ils exploitent, en plus de la logique, des valeurs de similarité syntaxique (Logic-based Fail et Nearest-neighbor). Les degrés d'appariement sont définis comme le présente la Figure 3.6.

Les degrés d'appariement de OWLS-MX sont ordonnés en tenant compte de la tolérance d'appariement sémantique, les auteurs de l'approche précisent que plus les données en outputs sont génériques par rapport à celles requises, plus l'appariement sémantique est tolérant. Ainsi, proposent-ils l'ordre suivant, allant du plus haut au plus bas degré :

Exact ; Plug-In ; Subsumes ; Subsumed-By ; Logic-based Fail ; Nearest-neighbor ; Fail

$LSC(C)$ the set of least specific concepts (direct children) C' of C, i.e. C' is immediate sub-concept of C
$LGC(C)$ the set of least generic concepts (direct parents) C' of C, i.e., C' is immediate super-concept of C
$Sim_{IR}(A, B) \in [0, 1]$ the numeric degree of syntactic similarity between strings A and B according to chosen IR metric
$\alpha \in [0, 1]$ given syntactic similarity threshold
$\doteq$ and $\dot{\geq}$ denote terminological concept equivalence and subsumption, respectively.

Exact match	Service S EXACTLY matches request R $\Leftrightarrow \forall$ IN$_S$ $\exists$ IN$_R$: IN$_S$ $\doteq$ IN$_R$ $\wedge$ $\forall$ OUT$_R$ $\exists$ OUT$_S$: OUT$_R$ $\doteq$ OUT$_S$.
Plug-in match	Service S PLUGS INTO request R $\Leftrightarrow \forall$ IN$_S$ $\exists$ IN$_R$: IN$_S$ $\dot{\geq}$ IN$_R$ $\wedge \forall$ OUT$_R$ $\exists$ OUT$_S$: OUT$_S \in$ LSC(OUT$_R$).
Subsumes match	Request R SUBSUMES service S $\Leftrightarrow \forall$ IN$_S$ $\exists$ IN$_R$: IN$_S$ $\dot{\geq}$ IN$_R$ $\wedge$ $\forall$ OUT$_R$ $\exists$ OUT$_S$: OUT$_R$ $\dot{\geq}$ OUT$_S$.
Subsumed-by match	Request R is SUBSUMED BY service S $\Leftrightarrow \forall$ IN$_S$ $\exists$ IN$_R$: IN$_S$ $\dot{\geq}$ IN$_R$ $\wedge$ $\forall$ OUT$_R$ $\exists$ OUT$_S$: (OUT$_S \doteq$ OUT$_R \vee$ OUT$_S \in$ LGC(OUT$_R$)) $\wedge$ SIM$_{IR}$(S, R) $\geq \alpha$.
Logic-based fail	Service S fails to match with request R according to the above logic-based semantic filter criteria.
Nearest-neighbor match	Service S is NEAREST NEIGHBOR of request R $\Leftrightarrow \forall$ IN$_S$ $\exists$ IN$_R$: IN$_S \geq$ IN$_R \wedge \forall$ OUT$_R$ $\exists$ OUT$_S$: OUT$_R \dot{\geq}$ OUT$_S$ $\vee$ SIM$_{IR}$(S, R) $\geq \alpha$.
Fail	Service S does not match with request R according to any of the above filters.

FIGURE 3.6 – Les degrés d'appariement dans OWLS-MX [45].

La Figure 3.6 présente les définitions formelles des degrés d'appariement pour la découverte dans l'approche OWLS-MX.

L'algorithme d'appariement reçoit comme requête un service OWL-S et retourne un ensemble ordonné de services jugés pertinents. A chaque service sont associés un degré d'appariement et une valeur de similarité syntaxique par rapport à la requête. L'utilisateur spécifie le degré d'appariement souhaité et le seuil de la valeur de similarité syntaxique.

L'algorithme ne calcule pas uniquement l'appariement logique des inputs et des outputs mais il calcule en parallèle les similarités syntaxiques entre les concepts des inputs et des outputs. L'échec d'un appariement logique au niveau du raisonneur logique de OWLS-MX est toléré si et seulement si la similarité syntaxique correspondante entre le concept du service en cours et de la requête reste supérieure au seuil de similarité souhaité.

Les auteurs de l'approche ont implémenté différentes variantes de l'algorithme générique de OWLS-MX : OWLS-M1, OWLS-M2, OWLS-M3 et OWLS-M4 [45]. Chacune utilise les mêmes degrés logiques mais des métriques de similarité syntaxique différentes. Une autre variante OWLS-M0 ne réalise que de l'appariement sémantique logique.

– OWLS-M0, les degrés d'appariement logiques Exact, Plugin et Subsumes sont appliqués comme précédemment définis, alors que le degré hybride Subsumed-By est utilisé sans vérifier aucune contrainte de similarité syntaxique.

– Les variantes hybrides OWLS-M1, OWLS-M2, OWLS-M3 et OWLS-M4 calculent la valeur de similarité syntaxique en se basant respectivement sur les mesures suivantes : LOI (Loss-Of-Information), le cœfficient ExtJac (extended Jacquard [31]), la valeur Cos (Cosine [27, 95]) et la mesure de divergence JS (Jensen-Shannon [26]).

3.4.3 OWLS-iMatcher2

OWLS-iMatcher2 [40, 38, 39] est une approche hybride de découverte dont la composante logique se base sur un appariement entre les concepts d'inputs/outputs et la composante non-logique utilise un appariement basé sur la similarité textuelle des noms et des signatures des services (SimPack [6, 7]). L'évaluation d'appariement utilise une valeur binaire de pertinence sémantique ainsi que les différentes valeurs de métriques de similarité telles que : Bi-Gram [38] et les mesures de Levenshtein [62, 29, 63], Monge-Elkan [71, 38] et Jaro [93, 38].

L'approche transforme la description structurée du profil du service en un vecteur pondéré de mots clés. Ce vecteur n'inclut pas que les noms utilisés mais aussi des termes qui en dérivent. L'algorithme d'appariement de OWLS-iMatcher2, commence par calculer les valeurs de similarités syntaxiques entre une requête donnée et tous les services disponibles, ensuite, il utilise un modèle mathématique de régression pour prédire l'agrégation d'appariement pour chaque service. Les résultats sont renvoyés, à l'utilisateur, en ordre décroissant d'appariement. L'utilisateur évalue la liste des services ordonnés. Pour cela, le OWLS-iMatcher2 lui offre deux composants : le premier permettant l'évaluation statistique des résultats retournés et le deuxième affichant une visualisation graphique.

Les évaluations expérimentales de OWLS-iMatcher2 ont été réalisées sur la collection de test OWLS-TC2.1 [1]. Les tests ont été réalisés sur une même base de services contenant à la fois les requêtes et leurs propres réponses, d'où un ensemble de résultats de tests influencés par cette stratégie d'évaluation de l'approche. En effet, le fait de concevoir à l'avance des ensembles contenant un nombre donné de services destinés à être les réponses exactes à chacune des requêtes rend moins le profil de la base de test moins réaliste. Ceci diminue le nombre de services considérés comme des faux positifs, ce qui dope la précision. Et cela favorise aussi la dcouverte des services qui répondent à la requête, ce qui dope le rappel.

3.4.4 FC-Match

L'approche hybride FC-Match (Functional Comparison) [9] se base sur un appariement logique et une similarité syntaxique. Les concepts du service et de la requête sont spécifiés en OWL-DL. Soit S un concept de service défini comme étant une conjonction d'expressions qualifiées de rôles. Chaque rôle correspond à un paramètre du profil sélectionné, le rôle est décrit par des concepts. Soit *C1* le concept associé au paramètre *catégorie* du profil de S, soit *C2* le concept associé au paramètre *opération*, soit *C3* le concept associé au paramètre *input*, soit *C4* le concept associé au paramètre *output*. Le concept global associé au profil du service S se note comme suit : S = ?hasCategory(*C1*) ?hasOperation(*C2*) ?hasInput(*C3*) ?hasOutput(*C4*)

Les degrés d'appariement dans FC-Match [9] sont calculés en agrégeant deux éléments :
– la combinaison des valeurs d'appariement basé sur la subsomption logique des concepts entre les deux profils (service et requête)

1. http ://users.auth.gr/ gmeditsk/owls-tc_v2.2_rev_2.html

– la valeur du cœfficient de similarité Dice :
  – entre les termes des concepts apparents dans les profils
  – en respectant les relations terminologiques spécifiées dans le thesaurus WordNet [92].

L'agrégation en appariement sémantique inclut les paramètres fonctionnels et non-fonctionnels d'un profil OWL-S :

– les paramètres fonctionnels : les inputs et les outputs,
– les paramètres non-fonctionnels : à travers hasCategory et hasOperation.

A ma connaissance, FC-MATCH n'est pas encore évalué expérimentalement.

3.4.5 WSMO-MX

WSMO-MX [36] accepte en entrée des services spécifiés en WSML-MX [47]. L'approche d'appariement est une combinaison d'appariement sémantique hybride de OWLS-MX [45], d'appariement de graphes, orienté objet du DSD-Matchmaker [42], et d'appariement intentionnel de service de [37]. La spécificité de ce matchmaker est la notion de "*derivative*". Une *derivative* DT est un concept ordinaire T "*Type*" qui est défini dans une ontologie donnée en lui attachant des méta-informations. Ces informations permettent de savoir comment apparier T avec un autre type.

Les degrés d'appariement sont calculés par l'agrégation des valuations de quatre éléments :

– l'appariement des types ontologiques (la relation hiérarchique entre concepts logiques)
– l'appariement logique (basé sur des instances) des contraintes spécifiées en F-logic
– l'appariement des noms de relations
– la mesure de similarité syntaxique

Les degrés d'appariement, leur ordre, leur symbole et leur signification par rapport aux états de précondition et postcondition, sont tous présentés dans la Figure 3.7. Soient **G** le **G**oal du service (le service requis) et **W** le service **W**SMO (le service offert). Les degrés sont cités en ordre décroissant de leur valeur d'appariement, allant de l'équivalence à l'échec. L'équivalence est en haut des degrés, notée *equivalence*. Ensuite viennent les deux degrés *plugin* et *inverse-plugin* qui sont définis différemment par rapport à un état de précondition ou de postcondition. Puis, on retrouve le degré *intersection* suivi du degré *fuzzy similarity*. WSMO-MX identifie en avant dernier le degré *neutral* et en dernier ordre le degré *fail* qui interprète un échec total de l'appariement.

ordre	symbole	degré d'appariement	pré	post
1	$\equiv$	equivalence	$\mathcal{G} = \mathcal{W}$	
2	$\sqsubseteq$	plugin	$\mathcal{G} \subseteq \mathcal{W}$	$\mathcal{W} \subseteq \mathcal{G}$
3	$\sqsupseteq$	inverse-plugin	$\mathcal{G} \supseteq \mathcal{W}$	$\mathcal{W} \supseteq \mathcal{G}$
4	$\sqcap$	intersection	$\mathcal{G} \cap \mathcal{W} \neq \emptyset$	
5	$\sim$	fuzzy similarity	$\mathcal{G} \sim \mathcal{W}$	
6	$\circ$	neutral	*by derivative specific definition*	
7	$\perp$	disjunction (fail)	$\mathcal{G} \cap \mathcal{W} = \emptyset$	

FIGURE 3.7 – Les degrés d'appariement dans WSMO-MX [36].

WSMO-MX offre les *derivative*s types des relations de similarité suivantes :

– *Equivalent* : les types TW (le Type en service WSMO) et TG (le Type en *Goal* : Objectif du service) sont équivalents,
– *Sub* : TW est un sous-type de TG,
– *Super* : TG est un sous-type de TW,
– *Sibling* : les types ont un type parent commun direct,
– *Spouse* : les types ont un type descendant commun direct,

– *ComAnc* : les types ont un parent commun non direct,
– *ComDes* : les types ont un descendant commun non direct,
– *Relative* : il existe un chemin dans le graphe non dirigé de l'ontologie qui les relie.

stratégies de défauts	vecteur de valuations	
	$val_{pre,webservice}/val_{post,goal}$	$val_{post,webservice}/val_{pre,goal}$
	($\pi_{\equiv}$,$\pi_{\sqsubseteq}$,$\pi_{\sqsupseteq}$,$\pi_{\sqcap}$,$\pi_{\sim}$,$\pi_{\circ}$,$\pi_{\perp}$)	($\pi_{\equiv}$,$\pi_{\sqsubseteq}$,$\pi_{\sqsupseteq}$,$\pi_{\sqcap}$,$\pi_{\sim}$,$\pi_{\circ}$,$\pi_{\perp}$)
$assumeEquivalent$	(1 , 0 , 0 , 0 , 0 , 0 , 0)	(1 , 0 , 0 , 0 , 0 , 0 , 0)
none	(0 , 1 , 0 , 0 , 0 , 0 , 0)	(0 , 0 , 1 , 0 , 0 , 0 , 0)
$ignore$	(0 , 0 , 0 , 0 , 0 , 1 , 0)	(0 , 0 , 0 , 0 , 0 , 1 , 0)
$assumeFailed$	(0 , 0 , 0 , 0 , 0 , 0 , 1)	(0 , 0 , 0 , 0 , 0 , 0 , 1)

FIGURE 3.8 – Valuations d'appariement de relations avec les stratégies de défauts [36].

relation de similarité des types	vecteur de valuations	
	val_{pre}	val_{post}
	($\pi_{\equiv}$,$\pi_{\sqsubseteq}$,$\pi_{\sqsupseteq}$,$\pi_{\sqcap}$,$\pi_{\sim}$,$\pi_{\circ}$,$\pi_{\perp}$)	($\pi_{\equiv}$,$\pi_{\sqsubseteq}$,$\pi_{\sqsupseteq}$,$\pi_{\sqcap}$,$\pi_{\sim}$,$\pi_{\circ}$,$\pi_{\perp}$)
$equivalent$	(1 , 0 , 0 , 0 , 0 , 0 , 0)	(1 , 0 , 0 , 0 , 0 , 0 , 0)
sub	(0 , 0 , 1 , 0 , 0 , 0 , 0)	(0 , 1 , 0 , 0 , 0 , 0 , 0)
$super$	(0 , 1 , 0 , 0 , 0 , 0 , 0)	(0 , 0 , 1 , 0 , 0 , 0 , 0)
$sibling$	(0 , 0 , 0 , 1 , 0 , 0 , 0)	(0 , 0 , 0 , 1 , 0 , 0 , 0)
$comAnc$	(0 , 0 , 0 , 1 , 0 , 0 , 0)	(0 , 0 , 0 , 1 , 0 , 0 , 0)
$spouse$	(0 , 0 , 0 , 0 , 1 , 0 , 0)	(0 , 0 , 0 , 0 , 1 , 0 , 0)
$comDes$	(0 , 0 , 0 , 0 , 1 , 0 , 0)	(0 , 0 , 0 , 0 , 1 , 0 , 0)
$relative$	(0 , 0 , 0 , 0 , 1 , 0 , 0)	(0 , 0 , 0 , 0 , 1 , 0 , 0)

FIGURE 3.9 – Valuations d'appariement pour relations de similarité des types [36].

WSMO-MX sélectionne les états de précondition et de postcondition de chaque service offert à partir de sa base de connaissances, ensuite il les apparie un par un avec les états du service requis. Dans le cas où il n'existe pas de préconditions, le résultat d'appariement de cet élément est fixé à *Equivalent* par défaut [36]. Dans ces cas de défauts (absences) d'annotations, WSMO-MX propose d'utiliser des stratégies spécifiques appelées "stratégies de défauts" (voir la Figure 3.8).

Après le traitement des résultats, l'algorithme d'appariement renvoie un vecteur de valuations d'appariements agrégées, avec des annotations sur les résultats du processus d'appariement. Ces annotations sont considérées comme une sorte de retour explicatif à l'utilisateur. Ce retour essaie, en cas de résultats d'appariement insuffisants, de faciliter le raffinement itératif de l'objectif *Goal* de la part de l'utilisateur. Pour calculer les degrés d'appariement sémantique, l'algorithme utilise des valeurs provenant des trois tables que nous présentons ici.

La table de la Figure 3.8 présente les valuations d'appariement de relations en tenant compte des stratégies spécifiques au cas de défauts. Il existe quatre cas de défauts :

– *assumeEquivalent* : le cas d'équivalence par défaut,
– *none* : le cas du nul,
– *ignore* : le cas où on ignore la relation de similarité,
– *assumeFailed* : le cas d'échec par défaut.

La table de la Figure 3.9 présente les valuations d'appariement de types. On retrouve les huit *derivative*s types des relations de similarité, à savoir : *Equivalent*, *Sub*, *Super*, *Sibling*, *Spouse*,

ComAnc, *ComDes* et *Relative*. Pour *Sub* et *Super*, uniquement les cœfficients des degrés d'appariement *plugin* et *inverse-plugin* s'intervertissent les valeurs respectivement en précondition et en postcondition. Les vecteurs sont les mêmes en état de précondition et en état de postcondition. Pour les autres relations, on a un même vecteur en état de précondition et en état de postcondition avec un unique cœfficient 1 et des zéros en reste du vecteur à chaque fois.

Le cœfficient 1 des vecteurs associés à la relation *Equivalent* correspond au degré d'appariement *equivalence*. Pour les relations *Sibling* et *ComAnc*, il correspond au degré *intersection*. Pour les relations *Spouse*, *ComDes* et *Relative*, il correspond au degré *fuzzy similarity*.

Le résultat d'appariement de type et toutes ces valuations sont agrégés selon un algorithme d'appariement sémantique.WSMO-MX était implémenté en utilisant le raisonneur *F-Logic* : *Onto-Broker*.

3.4.6 SAWSDL-MX

SAWSDL-MX [46] accepte en entrée des services spécifiés en SAWSDL [59]. Ce matchmaker est inspiré par les matchmakers OWLS-MX [45] et WSMO-MX [36]. La découverte utilise à la fois :

- une approche d'appariement logique basée sur le raisonnement par subsomption,
- une approche d'appariement syntaxique basée sur les techniques de recherche d'information.

Partant du fait qu'une description SAWSDL [24] est une extension d'une description WSDL, l'appariement recouvre les éléments de description suivants :

- interface si on découvre un service spécifié en WSDL 2.0 et portType si on découvre un service spécifié en WSDL 1.1,
- operation, input, output : pour WSDL 1.1 et WSDL 2.0.

Pour réaliser l'appariement des interfaces, le matchmaker effectue des appariements sur des graphes bipartis avec :

- des nœuds représentant des opérations d'un service,
- des arcs valués dont les valeurs sont calculées à partir des degrés d'appariement des opérations reliées par les arcs.

Pour réaliser l'appariement des opérations, le matchmaker utilise différentes techniques :

- basées sur une approche logique en utilisant les annotations sémantiques spécifiées par les attributs *modelReference* dans les éléments annotés,
- basées sur des approches syntaxiques : Loss-of-Information, Extended Jaccard [31], Cosine [27, 95] et Jensen-Shannon [26],
- basées sur des approches hybrides :
  - "Compensative" : le matchmaker utilise la similarité syntaxique lorsqu'aucun degré d'appariement logique ne peut être associé à un service,
  - "Intégrative" : pour remédier aux problèmes liés aux appariements faux positifs. Il ne faut pas uniquement prendre en compte la similarité syntaxique lorsque un appariement logique échoue, mais plutôt la considérer comme une contrainte conjonctive à chaque degré d'appariement logique.

Les degrés d'appariement dans une découverte de services avec SAWSDL-MX [46] peuvent être distingués en deux catégories :

– Les degrés d'appariement calculés par l'approche logique : *Exact, Plug-in, Subsumes et Subsumed-by*,
– Les degrés d'appariement calculés par l'approche hybride :*Subsumed-by* (s'il a besoin de calcul de similarité syntaxique additionnelle), *Nearest-neighbour* (pour compenser les appariements faux négatifs).

Au moment de l'agrégation des degrés d'appariement et du calcul d'appariement global entre deux services, l'approche opte pour la plus petite valeur. Ceci est appelé le principe d'agrégation "Valeur minimale" (notée Min). Exemple, dans la Figure 3.10, l'agrégation des deux degrés *Exact* et *Plug-in*, donne lieu au degré final d'appariement *Plug-in*. Nous démontrerons, ci-après dans l'examen des approches, que cette agrégation par valeur minimale présente des lacunes et ne permet pas parfois de déceler les meilleurs services Web pour une requête au moment de la découverte.

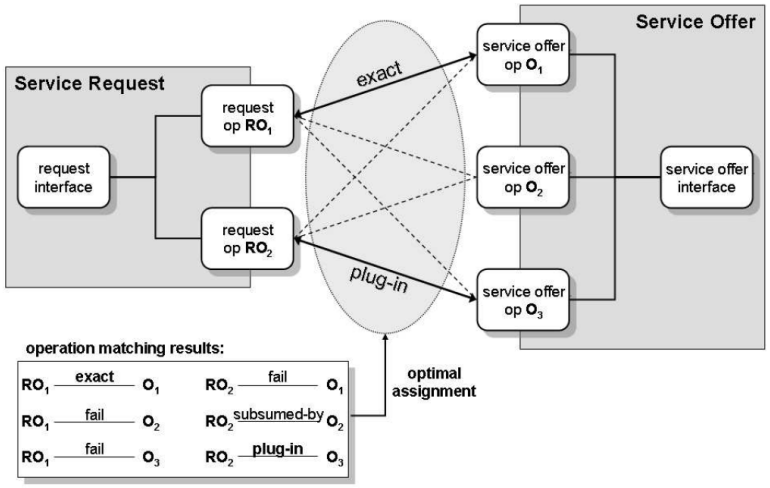


FIGURE 3.10 – Exemple d'application du principe d'agrégation Valeur minimale [46].

3.5 Examen des approches existantes

Après avoir présenté l'état de l'art de la découverte et d'appariement de services, j'examine, dans cette section, les divers travaux présentés. Pour ce faire, je commence par identifier les principaux critères de comparaison et j'utilise ces critères pour évaluer les approches étudiées.

3.5.1 Identification des critères d'évaluation

Les critères identifiés pour l'évaluation sont les suivants : dépendance vis-à-vis d'un type d'ontologie, éléments de description à apparier, technique d'appariement, degrés d'appariement, agrégation des résultats et intervention des utilisateurs.

Dépendance vis-à-vis d'un type d'ontologie

La plupart des approches utilisent des ontologies comme modèles sémantiques pour décrire leurs services Web. Le critère de dépendance vis-à-vis d'un type d'ontologies est important dans l'évaluation d'une approche. Une approche fermée est une approche qui dépend strictement d'un

type particulier d'ontologies et ne tolère pas d'autres types. Une approche ouverte est une approche qui ne dépend pas de types particuliers d'ontologies. Ceci offre plus de richesse dans la description, permet la réutilisation d'ontologies existantes, etc. J'identifie ainsi un premier critère : la dépendance vis-à-vis d'un type d'ontologie.

Éléments de description à apparier

La pertinence d'une approche de découverte dépend souvent des éléments à apparier entre la description du service Web requis et les descriptions des services Web offerts (e.g. dans l'extension SAWSDL du standard WSDL : les opérations, les inputs et les outputs). Examiner quels éléments, de toute une description, ont été gardés pour être apparier est primordial pour évaluer un certain aspect de la pertinence d'une approche : plus on réussit à apparier d'éléments des deux services, requis et offert, plus le résultat de découverte est pertinent. Toutefois, il faut souligner que ce n'est pas le grand nombre d'éléments appariés en lui-même qui importe, mais c'est plutôt le nombre d'éléments utiles qui fait la différence, vu l'information qu'ils peuvent apporter. Plus la portée de la découverte couvre d'éléments décrits distinctement, plus l'appariement est pertinent.

Technique d'appariement

Il existe un autre critère qui est plus ou moins lié avec le critère des éléments à apparier. Je m'intéresse ici à l'aspect purement technique du processus d'appariement. Ce processus peut procéder par des mécanismes syntaxiques, sémantiques, les deux à la fois ou par d'autres mécanismes. Il faut préciser que souvent, une technique d'appariement peut être appliquée différemment selon les éléments à apparier dans chaque approche de découverte. Prenons comme exemple les approches sémantiques. Certaines apparient simplement les concepts fonctionnels des inputs/outputs et certaines autres apparient en plus des concepts décrivant des aspects techniques fonctionnels et/ou non-fonctionnels. Plus la technique est complexe, plus cela coûte cher en termes de ressources aux approches, d'où l'intérêt d'identifier l'examen des techniques d'appariement qu'elles proposent.

Degrés d'appariement

La pertinence d'une approche de découverte dépend directement de la manière avec laquelle on évalue l'appariement entre les éléments de la description du service Web requis et les descriptions des services Web offerts. Le critère des "degrés d'appariement" est principal pour évaluer une grande partie de la pertinence d'une approche : plus on identifie précisément et fidèlement la similitude entre les éléments des deux services, requis et offert, plus le résultat de découverte est pertinent. Toutefois, il faut souligner que ce n'est pas le nombre de degrés qui importe, mais c'est plutôt le nombre de degrés significatifs qui fait la différence, vu l'information qu'ils peuvent apporter. Plus les degrés sont bien élaborés et distincts, plus l'interprétation de l'appariement et, par conséquent, la découverte sont pertinentes. Les degrés d'appariement peuvent être utilisés dans certaines approches pour évaluer l'appariement global de deux descriptions de services et dans d'autres approches pour évaluer l'appariement élémentaire au niveau des éléments de deux descriptions. Souvent, les approches présentent leurs degrés d'appariement sous forme d'échelle de valeurs signifiantes. Pour cela, l'élaboration de ces échelles, ou des degrés, est un facteur de succès ou d'échec des approches, d'où l'intérêt d'identifier l'examen des degrés d'appariement qu'elles proposent.

Agrégation des résultats

Un autre critère, plus au moins en relation avec le précédent, est l'agrégation des résultats d'appariements élémentaires (ou plus généralement partiaux). Comme son nom l'indique, je m'intéresse ici à la stratégie ou à l'aspect arithmétique de la manière avec laquelle les résultats partiaux d'appariements sont agrégés afin de calculer le degré final/global d'appariement pour la découverte. Il faut préciser que souvent, une agrégation des résultats d'appariements dépend du type des degrés d'appariement de chaque approche de découverte. Certaines approches utilisent des formules arithmétiques pour agréger des valeurs numériques, certaines d'autres utilisent des stratégies plus particulières telles que la valeur minimale, la moyenne voire même des algorithmes de tri et de sélection pour identifier le degré global.

Intervention des utilisateurs

Les motivations à mener une approche de découverte automatique de services Web sémantiques nous guide à éviter tout besoin d'entremise dans la procédure de découverte. L'intervention des utilisateurs à un stade du processus, présente toujours une limite dans une approche de découverte. L'exigence d'une assistance humaine est de plus en plus inacceptable dans un tel domaine (de plus en plus automatique). De plus, une interprétation humaine peut ne pas être assez pertinente, cela nécessite un bon niveau d'expertise. Une intervention humaine, même experte, peut coûter plus chère par exemple en termes de temps, qu'une approche automatique bien élaborée.

3.5.2 Évaluations des approches

Un résumé de résultats d'études des approches et leur comparaisons est affiché dans la Table 3.2.

Dépendance vis-à-vis d'un type d'ontologie

Comme il est indiqué dans la Table 3.2 au niveau de la troisième colonne, l'approche WSC [78] dépend de DAML (3e ligne). En effet, la requête et les services sont annotés par des concepts DAML et aucun appariement n'est identifié si la relation entre l'offre et la requête n'utilise pas des ontologies spécifiées en DAML et reconnues par l'annuaire de services publiés.

Sur la même colonne, on note que l'ensemble des travaux iMatcher1 [83], OWLS-M [32], OWLS-MX [45], OWLS-iMatcher2 [40] et FC-Match [9] dépendent de OWL-S. Quant à l'approche WSMO-MX [36], elle dépend de WSML. L'approche SAWSDL-MX [46] ne dépend d'aucun type d'ontologie. Les annotations sémantiques dans SAWSDL peuvent provenir de différents types d'ontologies.

Éléments à apparier

Comme il est indiqué dans la Table 3.2 au niveau de la quatrième colonne, dans les travaux WSC [78], OWLS-M [32] et OWLS-MX [45], l'appariement est basé principalement sur les inputs/outputs. Ce choix représente une limite pour ces approches. En fait, une découverte sémantique est ramenée de préférence à apparier divers éléments fonctionnels surtout lorsque plusieurs services se ressemblent au niveau des inputs/outputs. En effet, rien ne peut prouver que les méthodes offertes par un tel service correspondent aux méthodes requises par la requête. Souvent, les inputs/outputs ne sont que des paramètres pour des opérations. Et si l'appariement des opérations ne confirment pas que le service offert met bien à la disposition de la requête les

bonnes méthodes requises, alors il peut se révéler que les inputs/outputs ne sont pas suffisant pour apparier deux descriptions de services.

Quelques travaux tels que ALS [37] et GR [28] (lignes 5 et 6) ont essayé de proposer des solutions dans ce sens en ajoutant l'appariement autour de l'élément service lui-même (objectif, capacités, etc). SAWSDL-MX [46] ajoute, en plus des inputs/outputs, l'élément opération (Table 3.2, ligne 11).

Il est clair qu'une découverte limitée uniquement au niveau des inputs et des outputs reste insatisfaisante vue la richesse d'une description fonctionnelle. En fait, on peut toujours filtrer parmi les services retournés en faisant l'appariement uniquement sur leurs inputs/outputs, que ce soit syntaxiquement ou sémantiquement (comme dans [32], [45] et [23]). Il est évident que pour un ensemble de services offerts distincts mais proches en termes d'inputs/outputs, on doit trouver les informations distinctives à d'autres niveaux de leur description. L'approche WSMO-MX [36] est une approche d'appariement hétérogène et complexe. L'appariement pour la découverte se fait à plusieurs niveaux : il y a un appariement de types (définis par des concepts logiques), un appariement de contraintes logiques et un appariement de noms de relations (Section 3.4.5).

Techniques d'appariement

En regardant la cinquième colonne de la Table 3.2, je remarque que la plupart des approches sont basées sur une technique d'appariement soit syntaxique, soit sémantique, soit un mélange des deux. Par ailleurs, je remarque que d'une façon générale que les techniques non-logiques restent fortement liées aux mécanismes de (1) recherche d'information et (2) d'appariement de graphes. iMatcher1 [83] est un exemple pour le premier type de mécanisme (voir la Table 3.2, première ligne). DSD-Matchmaker [42] est un exemple pour le deuxième type de mécanisme (voir la Table 3.2, deuxième ligne). Toute évolution de ce type d'approches de découverte continue souvent à avancer dans le sens d'optimisations des algorithmes pour le calcul des similarités, pour le parcours et l'appariement de graphes, ou pour l'évaluation de formules numériques et statistiques.

Par ailleurs, des auteurs proposent les approches hybrides et se justifient qu'avec un faible coût supplémentaire, ils peuvent remédier à quelques limites de leurs approches purement logiques, comme dans l'approche SAWSDL-MX (Section 3.4.6). Mais ceci dépend tout de même des mécanismes logiques de chaque approche : souvent il n'y a pas besoin d'avoir recours à des techniques non-logiques si les techniques logiques sont bien fondées et suffisantes en termes de qualité et de résultats.

La principale critique autour de l'approche SAWSDL-MX en termes de pertinence des techniques d'appariement est qu'il n'y a aucune mention explicite des notions de précondition et effet. Ceci aurait permis à l'appariement et à la découverte d'éviter de sélectionner dans les résultats des services faux-positifs (considérés par l'algorithme comme bons alors que réellement ils ne répondent pas à la requête). Par exemple, si une condition est fournie comme étant une précondition du service alors que l'utilisateur la requiert comme effet, alors le service n'est pas bon car le matchmaker a simplement reconnu le concept de la condition mais pas sa bonne sémantique (sa bonne signification). En fait, le SAWSDL-MX apparie toujours de la même manière, par exemple, les annotations sémantiques sur les préconditions du service et les annotations sémantiques sur les résultats du service. Si on continue à spécifier les services en SAWSDL, les mêmes problèmes persistent pour les annotations sémantiques sur les contraintes fonctionnelles ou les propriétés non fonctionnelles, telles que le contexte, la QoS et les préférences utilisateurs.

WSMO-MX (Section 3.4.5), une approche hybride qui s'intéresse à découvrir des services Web décrits en WSML. En plus d'appariement de types définis par des concepts d'une ontologie, le matchmaker de WSMO-MX effectue un appariement de contraintes logiques, un appariement de noms de relation et un appariement de mesures de similarité syntaxique.

Les deux approches WSC [87] et une autre approche qui lui est proche proposée dans [78] se limitent à apparier une sémantique minimale apportée par quelques concepts logiques annotant leurs inputs/outputs. D'autres approches ajoutent une sémantique des services Web pour spécifier la nature de l'annotation elle-même. Mais ces approches se limitent aux concepts de "précondition" et "effet" des services Web (tel que dans [88]).

Dans la Table 3.2, au niveau de la colonne "technique d'appariement", un IOPE-matching caractérise les approches [32, 37, 55] et [45] (lignes 4 à 7).

Degrés d'appariement

En analysant les approches présentées dans les sections 3.2, 3.3 et 3.4, j'ai remarqué que les degrés d'appariement dans certains approches de découverte sont insuffisants pour décortiquer les différents cas d'appariement, comme dans OWLS-M [32], [23] et [78]. Dans certaines autres approches, les définitions des degrés ne sont pas assez claires pour dissocier les cas d'appariement, comme dans les travaux [37] et [55]. Au niveau de la sixième colonne de la Table 3.2, je cite les différentes propositions d'échelles de degrés d'appariement pour chaque approche (bien évidemment si elle en propose).

Dans OWLS-M (Table 3.2, ligne 4), j'identifie un seul cas d'appariement de relation hiérarchique à travers le degré *SUBSUMES*. Sinon, l'algorithme de découverte proposé renvoie soit le degré d'appariement *UNKNOWN* quand l'appariement ne peut pas être jugé, soit le degré d'appariement *FAIL* en cas d'échec. En fait, le choix des auteurs représente une limite de leur approche, vus les différents autres cas spécifiques de relations hiérarchiques souvent identifiées par d'autres travaux, par exemple, le degré *Plug-in* dans WSC (ligne 3) ou *Subsumed-by* dans OWLS-MX (ligne 7).

Au contraire des approches qui ont proposé des degrés plus précis pour prendre en compte le plus possible de relations hiérarchiques, la découverte dans OWLS-M risque souvent d'affronter des situations d'indécision ou d'échec sur des cas d'appariement jugés significatifs par d'autres approches. La preuve est donnée par l'exemple suivant : supposons que nous cherchons un service qui, à partir d'un numéro de série d'un téléphone mobile en entrée, il est capable de nous renvoyer le nom de son fabricant. En effet, un service qui renvoie le nom de fabricant en acceptant en entrée un numéro de série de tout type d'appareil électronique, satisferait toujours notre requête, alors que dans le cas contraire, un service qui n'accepte que des numéros de série de téléphones mobiles, aurait du mal à satisfaire toutes les requêtes requérant en entrée des numéros de série d'autres types d'appareil électronique. En conclusion, des services offerts, ayant des inputs annotés avec des concepts plus spécifiques que les concepts d'inputs requis, ne peuvent pas toujours satisfaire l'utilisateur ; alors que des services offerts, ayant des inputs annotés avec des concepts plus génériques que les concepts d'inputs requis, sont aussi satisfaisants que des services offerts ayant des inputs avec des concepts équivalents aux concepts d'inputs requis. OWLS-M [32] n'est pas sensible à cet aspect vu qu'il ne propose qu'un de degré d'appariement pour le cas le moins satisfaisant.

Agrégation des résultats

L'approche WSMO-MX (Section 3.4.5) propose une agrégation un peu plus élaborée qui se distingue des autres approches. Elle agrège des évaluations d'appariements beaucoup plus hétérogènes. En fait, en plus des appariements sur les types définis par des concepts d'ontologies, le matchmaker WSMO-MX effectue un appariement de contraintes logiques, un appariement de noms de relation et un appariement de mesures de similarité syntaxique. Les évaluations des appariements sont enregistrées sous forme de vecteurs qui sont agrégés pour obtenir le résultat final d'appariement entre deux services. Le résultat est le vecteur somme des vecteurs enregistrés.

L'algorithme du matchmaker OWLS-MX (Section 3.4.2) commence par apparier les profils des services, puis les inputs, ensuite les outputs et enfin les propriétés non fonctionnelles. Une étape de ces quatre n'est plus nécessaire si la précédente échoue. En réalité, on peut toujours trouver un service publié avec, par exemple, une majorité d'inputs qui s'apparient exactement et quelques autres qui s'apparient partiellement et sans que cela n'empêche la bonne exécution du service. Dans ce cas de figure, un échec d'appariement pénalise un service potentiellement utile.

SAWSDL-MX [46] propose de ne garder, comme valeur finale d'agrégation, que le pire degré résultant des appariements des éléments de la description requise, c'est l'approche par "valeur minimale". J'ai repéré un inconvénient majeur dans cette approche. La Table 3.1 présente un exemple de démonstration pour cet inconvénient. La Table spécifie le degré d'appariement de chaque élément d'un service offert par rapport à l'élément correspondant dans le service requis. L'échelle d'appariement est de 0 à 5 avec : 0 signifie "ne s'apparie pas", 5 signifie "s'apparie exactement", 3 signifie "s'apparie partiellement" ; par exemple dans notre cas, le concept offert est un sous-concept du concept requis.

TABLE 3.1 – Exemples d'agrégations avec l'approche par valeur minimale.

Services	Operation1	Input1	Output1	Operation2	Input2	Output2	Valeur Minimale
1er Service	5	5	5	3	5	5	3
2ème Service	3	3	5	3	5	3	3

Pour évaluer l'efficacité de l'approche "Valeur minimale", nous appliquons le principe de l'agrégation sur les degrés d'appariement élémentaires des deux services et nous comparons les résultats.

– Dans la ligne 1 : l'agrégation sélectionne parmi { 5 , 5 , 5 , 3 , 5 , 5 } le plus petit degré : 3,
– Dans la ligne 2 : l'agrégation sélectionne parmi { 3 , 3 , 5 , 3 , 5 , 3 } le plus petit degré : 3,

Donc, finalement les deux services publiés sont considérés au même niveau si on ne se fie qu'aux résultats finaux. Mais il est clair que le premier service satisfait beaucoup mieux la requête utilisateur que le deuxième service. La différence des degrés d'appariement au niveau des deux opérations est importante mais l'approche d'agrégation ne le décèle pas.

Intervention des utilisateurs

Les types d'intervention utilisateur sont spécifiés au niveau de la dernière colonne de la Table 3.2. Bien que considérée souvent comme une limite surtout quand elle coûte cher en termes de temps, l'intervention de l'utilisateur est exigée dans les approches iMatcher1 (ligne 1), WSC (ligne 3), OWLS-MX (ligne 7), FC-Match (ligne 9), WSMO-MX (ligne 10) et SAWSDL-MX (ligne 11).

Dans ces approches, deux interventions sont requises :
– Le choix du seuil à partir duquel un appariement est jugé réussi par l'utilisateur pour sa requête,
– La sélection du service jugé par l'utilisateur comme le plus proche de sa requête. La sélection est précédée d'une vérification humaine des résultats de la découverte.

Il est clair que de tels types d'interventions ne conviennent pas aux cas où la découverte se veut automatique ou quand un intervenant humain n'est pas disponible.

L'approche DSD-Matchmaker (ligne 2) exige seulement que l'utilisateur sélectionne à la fin le service qui lui correspond parmi une liste de services candidats. L'approche OWLS-M (ligne 4) nécessite que l'utilisateur choisisse des contraintes d'appariement et sélectionne à la fin le service qui lui correspond parmi une liste de services candidats.

Récapitulatif

Concernant les matchmakers logiques et hybrides, je précise que leurs composantes logiques sont conçues dans la quasi-totalité des travaux pour apparier uniquement une requête et un service utilisant une même ontologie. C'est-à-dire qu'il n'y a pas de degré d'appariement si les concepts requis et offert proviennent d'ontologies différentes. Rares sont les solutions qui traitent de cette problématique (voir les travaux présentés dans [64]). Ceci se résume à trouver une réponse à la question suivante : "Que peut on faire quand un fournisseur publie un service qui théoriquement s'apparie avec une requête mais que les deux peuvent être décrits par des ontologies tout à fait différentes ?". Vu que la réponse dépasse le domaine des services Web et touche à l'appariement d'ontologies, je choisis de rejoindre l'ensemble des travaux qui partent de l'hypothèse que l'appariement se fait entre deux concepts d'une même ontologie. Ceci n'empêche pas que la description peut exploiter plusieurs ontologies, mais je tiens à préciser que toute problématique liée à la construction et l'évolution des ontologies est en dehors du champ de mes travaux.

Finalement, je remarque que les approches d'appariement les plus citées et les plus élaborées sont celles basées sur la logique (pure ou hybride) tels que SAWSDL-MX , OWLS-MX et WSMO-MX . Le plus important est que ces mêmes travaux utilisent les langages de description de services les plus connus et répondus, à savoir SAWSDL, OWL-S et WSMO. Un intérêt pour SAWSDL que j'ai dégagé dans la partie description, se confirme vu que le travail sur SAWSDL-MX se base et résume les choix faits dans WSMO-MX et OWLS-MX. Une éventuelle nouvelle approche doit, à la fois, tirer profit des avantages de ses prédécesseurs et combler leurs lacunes. Un IOPE-matching amélioré/avancé permettra à une approche logique d'améliorer les performances des algorithmes de découverte, grâce à une couverture exhaustive de l'aspect fonctionnel et technique d'une description de service. Je propose d'ajouter aux valuations d'appariement des inputs/outputs, les valuations d'appariement de deux importants éléments fonctionnels : "opération" et "interface". Je propose aussi de ne pas se limiter aux concepts "précondition" et "effet".

L'approche d'appariement que je propose doit mettre en œuvre des degrés d'appariement plus précis et pertinents que ceux déjà proposés et un algorithme d'appariement plus performant au niveau précision et rappel. En effet, je propose un paradigme d'appariement qui exploite d'avantage de concepts de services Web, provenant des ontologies de service et méta-modèles les plus répandus (standards ou largement utilisés). Je propose que ces concepts soient enrichis et fusionnés dans une seule ontologie et ainsi je pourrais par exemple apparier les descriptions de service à travers des concepts tels que postcondition, capability, goal, result, etc. Ce qui représente un grand atout par rapport aux autres approches, au niveau de la précision d'appariement et de l'efficacité de la découverte.

L'examen de l'état de l'art de la découverte et d'appariement de services démontre le besoin d'une découverte plus précise basée sur une expressivité sémantique plus explicite.

3.6 Conclusion

Dans ce chapitre, j'ai présenté l'état de l'art de la découverte et d'appariement de services. Ensuite, j'ai examiné les différents travaux présentés selon leurs approches. Pour examiner ces approches, j'ai commencé par identifier les critères les caractérisant. Finalement, j'ai utilisé ces critères pour évaluer les approches étudiés. La discussion autour de l'état de l'art de la découverte et d'appariement de services a démontré le besoin d'une découverte plus précise basée sur une expressivité sémantique plus explicite.

TABLE 3.2 – Table comparative des travaux autour de la découverte de services Web.

Travaux étudiés	**Catégorie de l'approche**	**Dépendance vis-à-vis de l'ontologie de service**	**Niveaux d'appariement**	**Techniques d'appariement**	**Degrés d'appariement**	**Agrégation des résultats**	**Intervention des utilisateurs**
iMatcher1 [83]	Non-logique	OWL-S	Eléments du profil du service	Syntaxique	Pas d'échelle de degrés	Numérique (formule)	Choix du seuil et Sélection du service
DSD-Matchmaker [42]	Non-logique	DSD	Noms des nœuds et Noms des concepts	Appariement de graphes	Pas d'échelle de degrés	Numérique (formule)	Sélection du service
WSC [78]	Logique	DAML	Input et Output	IO-matching (Sémantique)	Échelle : *Exact, Plug-in, Subsume et Fail*	Algorithme de tri	Choix du seuil et Sélection du service
OWLS-M [32]	Logique	OWL-S	Input et Output	IOPE-matching (Sémantique)	Échelle : *Equivalent, Subsume, Unknown et Fail*	Numérique et manuelle	Choix de contraintes d'appariement et Sélection du service
ALS [37]	Logique	Pas d'ontologie : Goals, Services abstraits, Capacités de service	Input, Output et Goal de service	IOPE-matching (Sémantique)	Échelle : *Match, PossMatch, ParMatch, PossParMatch et NoMatch*	Non spécifiée	Non spécifiée
Graded Relevance [55]	Logique	Pas d'ontologie : Goals, Services abstraits, Capacités de service	Input, Output et Goal de service	IOPE-matching (Sémantique)	Échelle : *Match, PossMatch, ParMatch, PossParMatch, NoMatch, Relation-Match et Excess-Match*	Non spécifiée	Non spécifiée
OWLS-MX [45]	Hybride	OWL-S	Input et Output	IOPE-matching (Sémantique et Syntaxique)	Échelle : *Equivalent, Subsume, Subsumed-by, Logic-based fail, Nearest-neighbor et Fail*	Numérique (formule) et Algorithme de tri	Choix du seuil et Sélection du service
OWLS-iMatcher2 [40]	Hybride	OWL-S	Input et Output	IO-matching (Sémantique et Syntaxique)	Pas d'échelle de degrés	Numérique (formule) et Algorithme de tri	Non spécifiée
FC-Match [9]	Hybride	OWL-S	Input, Output et Propriétés non-fonctionnelles	IO-matching et Non-fonctionnel (Sémantique et Syntaxique)	Pas d'échelle de degrés	Numérique (formule) et Algorithme de tri	Choix du seuil et Sélection du service
WSMO-MX [36]	Hybride	WSML	Goal de service	Goal-matching (Sémantique et Syntaxique)	Échelle : *Equivalence, Plug-in, Inverse-plug-in, Intersection, Fuzzy similarity, Neutral, Disjunction (fail)*	Moyenne de vecteurs des degrés d'appariement	Choix du seuil et Sélection du service
SAWSDL-MX [46]	Hybride	Pas d'ontologie	Input, Output et Opération	IOPE-matching (Sémantique et Syntaxique)	Échelle : *Equivalent, Subsume, Subsumed-by, Nearest-neighbor et Fail*	Degré minimal	Choix du seuil et Sélection du service

Troisième partie

Contributions

Chapitre 4

La description de service YASAWSDL

Sommaire

4.1 Introduction

Les services Web sont basés sur des standards tels que WSDL pour la description de leurs propriétés fonctionnelles (interfaces, opérations, inputs, outputs, etc). Le manque de sémantique dans WSDL empêche l'appariement et la découverte automatiques, ainsi que l'invocation et la composition automatiques. Certains efforts tels que SAWSDL proposent une annotation sémantique aux descriptions de services Web standard WSDL afin de répondre à ce manque. Mais comme il est indiqué dans le chapitre 1 et le chapitre 3, les approches étudiées dans l'état de l'art, y compris SAWSDL, présentent encore soit des lacunes à combler soit des inconvénients à éviter.

Dans mes travaux, je m'intéresse à étendre les approches existantes en termes de description et de découverte de services Web sémantiques. Ce chapitre présente mon approche Yet Another Semantic Annotation for WSDL (YASAWSDL, en plus court YASA), qui est une extension de SAWSDL. En fait, SAWSDL se limite à annoter à gros grain les descriptions de services Web avec des ontologies métiers et ne précise pas d'avantage la nature de relation entre les concepts logiques et les éléments fonctionnels annotés dans une description de service Web.

Le principe général de la contribution est l'intégration de la sémantique dans la description de service Web en se basant sur l'utilisation de deux types d'ontologies. Une première ontologie dite

"ontologie technique" contenant des concepts définissant des sémantiques -fonctionnelles et/ou non fonctionnelles- de services et une deuxième ontologie dite "ontologie de domaine" contenant des concepts définissant les sémantiques métiers des services. Ce principe permet à YASA d'augmenter non seulement son expressivité par rapport à WSDL et SAWSDL mais aussi par rapport à l'expressivité dans OWL-S et WSMO puisque mon approche vient s'inspirer des leurs, en tire les avantages et en comble les lacunes.

Avec ce chapitre, j'entame la présentation de mes contributions dans ce travail de recherche. Je commence ici par le langage de description sémantique de services Web YASA qui est à l'origine de toutes les autres contributions, à savoir, l'ontologie des services Web, l'appariement et la découverte sémantique et l'annuaire sémantique de services Web. Les deux dernières sont présentées dans les chapitres qui suivent.

Ce chapitre présente tout d'abord, dans la section 4.2 les motivations de cette contribution. La section 4.3 présente les principes du langage de description sémantique YASA, les avantages et l'utilisation de l'approche. Ensuite, l'Ontologie Technique des Services Web sémantiques OTES, utilisée par l'approche YASA, est présentée dans la section 4.4.

4.2 Motivations de la contribution

Les services Web, étant modulaires et auto-descriptifs, fournissent un modèle simple de programmation et de déploiement. La sémantique dans les services Web a permis pour certains services qui l'ont adoptée dans leur description, de faire évoluer leurs mécanismes de découverte au-delà des mécanismes syntaxiques habituels. C'est le cas pour WSDL [13]. Le manque de sémantique dans les travaux autour du standard des services Web WSDL rend difficile son utilisation efficace pour la découverte et la composition automatiques. Plusieurs approches essaient de combler ce manque en développant des modèles sémantiques pour la description des services Web. Parmi ces approches, je cite les plus utilisées : OWL-S [66], SAWSDL [24] et WSMO [21]. Ces approches utilisent des ontologies de domaines pour ajouter des concepts à leurs descriptions. Un concept sémantique est utilisé pour annoter une partie de la description du service Web par le concept métier correspondant. Les langages basés sur la sémantique tels que OWL-S et WSMO sont des "approches fermées" : d'une part, elles ne manipulent qu'un langage de spécification d'ontologie, par exemple OWL pour OWL-S et WSML pour WSMO [90]. D'autre part, elles ne spécifient qu'un ensemble défini mais très limité de concepts qui ne sont pas facilement extensibles.

L'ontologie de service utilisée dans WSMO propose des concepts décrivant des objectifs de haut niveau et une approche sémantique proche de OWL-S mais WSMO se focalise sur la médiation d'Objectifs (dits Goals) et la chorégraphie. Quant à OWL-S, il se focalise sur la modélisation de processus. OWL-S peut utiliser des concepts fonctionnels que WSMO n'utilise pas et vice-versa. Dans SAWSDL, il n'y a aucune mention explicite des concepts de précondition et d'effet qu'on peut trouver dans les ontologies de services OWL-S et WSMO. En plus, SAWSDL n'est pas dédié pour décrire le comportement de services Web qui peut être un élément essentiel pour l'invocation et la composition. Cependant, SAWSDL reste une approche indépendante du langage de représentation sémantique grâce à la séparation entre les mécanismes d'annotation sémantique et la représentation de la description sémantique. Sans cette séparation, les développeurs n'ont pas assez de flexibilité pour choisir leurs langages de représentation sémantique favoris,

ou pour réutiliser leurs propres ontologies techniques et annoter les descriptions avec diverses ontologies.

Afin d'améliorer la découverte de services, j'ai identifié dans la section 1.4 certains objectifs à réaliser autour de certains critères concernant la description des services Web sémantiques. La spécification SAWSDL propose d'utiliser un attribut noté "*modelReference*" dans les éléments des documents WSDL et XML Schema. Toutefois, SAWSDL ne définit la sémantique que pour les éléments XML *interface, operation, fault, element, complexType, simpleType et attribute* [24]. Il convient de préciser que la recommandation donnée concernant l'utilisation des attributs "*modelReference*" pour chacun de ces éléments va beaucoup plus dans le sens d'une suggestion que d'une définition/spécification [67]. Par exemple, souvent avec les interfaces, on mentionne que l'attribut "*modelReference*" peut être utilisé pour les classer dans un modèle particulier, préciser certains aspects du comportement du service Web ou d'autres définitions sémantiques, et de même pour les opérations. Par conséquent, lorsqu'une opération est annotée en utilisant plusieurs concepts provenant d'une ontologie, on n'est pas capable de différencier ces concepts ! Par exemple, quel concept signifie/annote la catégorie de l'interface ? Quel concept signifie/annote le comportement ? Quel concept signifie/annote la QoS ? Et ainsi de suite. Des constatations similaires peuvent être faites pour toute annotation sémantique d'un élément WSDL. Par conséquent, il est nécessaire d'être en mesure de différencier la description sémantique de tous les éléments des services. La différenciation des annotations sémantiques des éléments WSDL peut être utilisée principalement pour améliorer la découverte de services Web.

Au lieu d'utiliser un seul attribut (le "*modelReference*") pour associer (une ou) plusieurs propriétés sémantiques à un élément WSDL, je devrais enrichir le mécanisme de description afin de distinguer les propriétés. Ceci permet donc de différencier chaque propriété sémantique des autres. C'est cette distinction qui permet de baser une découverte de services Web sur une propriété sémantique spécifique et pas les autres (par exemple, découvrir des services Web en se basant sur leurs effets). La composition et l'invocation de services Web peuvent utiliser cette distinction pour se baser sur une propriété sémantique spécifique en particulier (par exemple, le comportement des services Web).

Dans cette contribution, je m'intéresse à la description des services Web sémantiques. L'idée est de proposer une description sémantique basée sur les standards de-facto pour les services Web, principalement WSDL ; et que cette description n'exige de changements ni au niveau de la norme des documents WSDL, ni au niveau de la norme des documents XML Schema [22]. La nouvelle description ne doit pas éventuellement exiger un changement dans la façon avec laquelle ces types de documents (WSDL, XML Schema) sont traités habituellement (par exemple en *Parsing*) ; par suite nous pouvons utiliser tous les outils déjà développés qui respecte le standard, nous réutilisons ainsi les outils de *Parsing* des documents ou les outils d'invocation de services. Pour atteindre cet objectif, je propose un langage de description sémantique de services Web qui étend SAWSDL et qui intègre la sémantique en se basant sur l'utilisation de deux types d'ontologies. Le premier type d'ontologies dit "technique" telles que les ontologies contenant des concepts définissant la sémantique fonctionnelle des services (exemples : interface, effet, etc.) et/ou des ontologies contenant des concepts non fonctionnels (exemples : QoS, types d'attributs sensibles au contexte, etc). Le deuxième type d'ontologies dit "métier", dit aussi "de domaine" contient des concepts définissant la sémantique métier des services (exemples : tourisme, santé, commerce, etc).

Après avoir présenté les motivations de la contribution en description de services Web, je présente dans la section suivante le langage de description sémantique YASA pour les services Web.

4.3 Le langage de description sémantique YASA

Dans cette section, je présente notre contribution à la description sémantique de services Web, à savoir le langage YASA. Je commence par introduire le principe du langage YASA, ensuite, je l'illustre à travers un exemple. Enfin, je présente ses avantages et certaines contraintes.

4.3.1 Principe

YASAWSDL [10] (dit aussi YASA) est un langage de description de services Web qui étend WSDL et SAWSDL afin d'améliorer l'approche standard et résoudre les problèmes d'ambiguïté relevées chez SAWSDL (voir section 5.3.3).

Par exemple, l'annotation d'une opération par un concept d'un domaine métier est une annotation à gros grain, puisque elle ne distingue pas une précondition, d'un effet ou d'un résultat d'une opération.

L'idée est d'étendre SAWSDL afin d'améliorer l'expressivité de la description des services. Cette approche permet dorénavant l'annotation d'un élément WSDL en utilisant des concepts techniques d'une ontologie de services Web. L'idée est de spécifier la nature en sémantique de services des concepts provenant d'ontologies de métier référencés et utilisés dans une description SAWSDL.

Ainsi le système d'annotation des descriptions de service Web YASA, propose-t-il que le nouvel attribut, nommé "*serviceConcept*", spécifie des concepts techniques d'une ontologie de service qui correspondent, un à un dans l'ordre de leur déclaration, à des concepts d'une ontologie d'un domaine métier référencés dans l'attribut "*modelReference*" de SAWSDL.

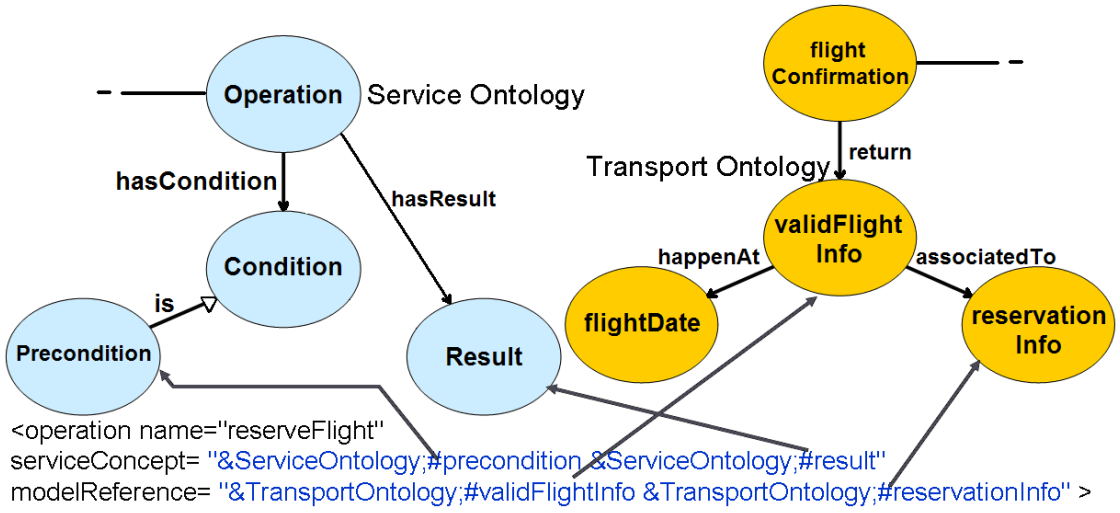


FIGURE 4.1 – Système d'annotation dans YASA.

Dans l'exemple de la Figure 4.1, j'utilise deux types d'ontologie. La première ontologie dite "ontologie technique" décrit la sémantique de plusieurs concepts des services Web (ex. "*precondition*" et "*effect*"). La deuxième ontologie dite "ontologie de domaine" est une ontologie de réservation de vols. L'exemple présente l'annotation sémantique d'une opération nommée "*reserveFlight*". L'attribut "*modelReference*" référence deux concepts de l'ontologie de domaine : "*validFlightInfo*" et "*reservationInfo*". L'attribut "*serviceConcept*" offre la capacité de distinguer les rôles joués par les deux concepts "*validFlightInfo*" et "*reservationInfo*", référencés par l'attribut "*modelReference*". Pour cela, l'attribut "*serviceConcept*" précise que ces deux concepts correspondent respectivement aux concepts de services : "*precondition*" et "*effect*". L'ordre est important, puisque on associe le premier concept technique "*precondition*" au premier concept de domaine "*validFlightInfo*", le second concept technique au second concept de domaine, et ainsi de suite.

YASA raffine la description métier grâce aux nouvelles annotations techniques. Ce raffinement renforce l'expressivité de la description sémantique. Il existe un autre avantage pour l'aspect technique. Dans YASA, je conçois une ontologie technique de services avec la possibilité de l'étendre par de nouveaux concepts en intégrant soit de nouveaux éléments de description soit des concepts plus précis. L'avantage offert par l'approche est que ces extensions et ces enrichissements n'auront aucun impact sur le système d'annotation. Il sera indépendant, d'une part, du langage et de l'ontologie assurant la représentation du domaine, et d'autre part, du langage et l'ontologie assurant la représentation de la sémantique métier des services Web. Cet aspect de YASA offrira aux développeurs plus de liberté sur le choix de leurs ontologies techniques et de leurs langages de représentation sémantique favoris, ceci leur permettra de réutiliser leurs ontologies techniques et d'annoter les descriptions avec plusieurs ontologies.

4.3.2 Exemple support de description sémantique YASA

Une agence immobilière gère différents biens sur plusieurs villes et plusieurs régions (maisons, appartements, salles des fêtes, garages, maisons de plage) qu'elle propose, à louer ou à vendre. Elle est composée de l'agence, équipée d'ordinateurs fixes, et de plusieurs agents mobiles, équipés de divers terminaux hétérogènes, et présents sur le terrain. Certains agents mobiles s'occupent des maisons, et d'autres s'occupent des appartements. L'agence met à la disposition des clients et de ses agents mobiles différents services Web pour consulter les différents types de biens, les localiser et s'y rendre s'ils veulent afin de les visiter. Certains services Web concernent la location, d'autres la vente et d'autres l'échange. Certains services fournissent une fonctionnalité d'un service de localisation avec le réseau GSM. L'exemple étudié ici est un cas d'utilisation "Localiser via GPS un appartement pour l'achat et avoir un plan de carte pour s'y rendre", défini comme suit : Un agent ou un client sur le terrain demande le service Web qui lui permet de trouver les biens de type "appartement" à vendre qui se situent dans le périmètre où il se trouve (en spécifiant la distance métrique maximale entre lui et un bien). L'agence immobilière offre sur les terminaux de ses agents ou ses clients le service Web adéquat à la sémantique spécifiée dans cette requête de découverte.

Les opérations sont définies comme suit :

- "getLocationGPS" renvoie la position actuelle de la personne demandant le service Web en termes de latitude et de longitude des coordonnées GPS.
- "getApartmentBuyList" récupère la liste des appartements proches et disponibles à l'achat.

– "getMap" récupère le plan de carte et les informations nécessaires pour pouvoir se rendre à l'appartement.

Un extrait de l'exemple de description d'un des services Web de l'agence immobilière est présenté dans la Figure 4.2.

Comme dans SAWSDL, une description contient quatre principales parties : types (lignes 11-14), interface(s) (lignes 16-62), binding(s) (lignes 64-65) et service (lignes 67-72). L'élément XML "*types*" définit le modèle de contenu des éléments XML "*element*" tel qu'une définition par un schéma XML global. L'élément "*interface*" est un ensemble d'éléments "*operation*". Et il décrit les messages qu'un service envoie et/ou reçoit. L'élément XML "*operation*" est une séquence de messages d'entrée "*input*" et de sortie "*output*" (lignes 15-16). L'élément XML "*binding*" décrit le format de message concret et le protocole de transmission qui peuvent être utilisés pour définir un "*endpoint*" du service, il définit les détails d'implémentation nécessaires pour accéder au service. L'élément "*service*" décrit un ensemble de "*endpoints*" à partir des quels une implémentation particulière déployée du service est fournie (ligne 69). L'élément XML "endpoint" est la localisation à partir de laquelle le service est accessible [24].

L'annotation sémantique de l'interface, appelée "*GeoLocApartBuyMapServiceInterface*" (ligne 16) est assurée par deux attributs : "*modelReference*" (ligne 23) et "*serviceConcept*" (ligne 25). Ils indiquent respectivement que à condition d'avoir le GPS, l'interface a pour but de localiser des appartements disponibles chez l'agence pour effectuer un achat. Tout est défini grâce à l'ontologie du domaine "agensimo" de l'agence immobilière d'une part (les concepts : terminalGPS, buy et apart) et l'ontologie technique de services d'autre part (les concepts Precondition, Goal et Effect).

Dans l'opération "*getLocationGPS*" (ligne 22), son attribut "*serviceConcept*" (ligne 25) fournit une liste de concepts de services auxquels correspondent, respectivement et dans le même ordre, une liste de références du modèle de l'ontologie de domaine de l'agence fournis par l'attribut "*modelReference*" (ligne 23 et 24).

L'opération a comme précondition ("*Precondition*") le concept métier "*GPSactive*" : qui vérifie que le GPS est disponible pour la localisation. L'opération a pour résultat ("*Result*") le concept métier "*GPSposition*" ce sont les coordonnées GPS renvoyées par le satellite pour la géolocalisation du client ou de l'agent. L'input est le signal GPS (lignes 26-29) et l'output est les coordonnées GPS (lignes 30-33).

Dans l'opération "*getApartmentBuyList*" (ligne 36), son attribut "*serviceConcept*" (ligne 39) fournit une liste de concepts de services auxquels correspondent, respectivement et dans le même ordre, une liste de références du modèle de l'ontologie de domaine de l'agnece fournis par l'attribut "*modelReference*" (ligne 37 et 38).

L'opération a comme précondition ("*Constraints*") le concept métier "*buy*" qui projette que le client ou l'agent cherche un bien pour l'achat. L'opération a pour résultat ("*Result*") le concept métier "*apart*" qui sont la liste d'appartements disponibles et proches de la position GPS du client ou de l'agent immobilier. L'input est la contrainte de localisation proche souhaitée de l'appartement à acheter (lignes 40-43) et l'output est les appartements proches et disponibles à vendre (lignes 44-47).

Dans l'opération "*getMap*" (ligne 50), son attribut "*serviceConcept*" (ligne 52) fournit un concept de services auxquel correspond une référence du modèle de l'ontologie de domaine de l'agence fourni par l'attribut "*modelReference*" (ligne 51).

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:description
  targetNamespace="http://www.telecom-sudparis.eu/GeoLocApartBuyMapService.wsdl20"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdl="http://www.w3.org/ns/wsdl"
  xmlns:tns="http://www.telecom-sudparis.eu/GeoLocApartBuyMapService/"
  xmlns:sawsdl="http://www.w3.org/ns/sawsdl"
  xmlns:yasawsdl="http://www-inf.it-sudparis.eu/SIMBAD/tools/YASA-R/ns/yasawsdl/"
  name="GeoLocApartBuyMapWebService">

  <wsdl:types>
    <xs:import    namespace="http://www.telecom-sudparis.eu/GeoLocApartBuyMapService.xsd"
            schemaLocation="http://www.telecom-sudparis.eu/GeoLocApartBuyMapService.xsd" />
  </wsdl:types>

  <wsdl:interface name="GeoLocApartBuyMapServiceInterface"
    sawsdl:modelReference="http://localhost/agensimo#terminalGPS
                           http://localhost/agensimo#buy
                           http://localhost/agensimo#apart"
    yasawsdl:serviceConcept="Precondition Goal Effect">

    <wsdl:operation name="getLocationGPS"
      sawsdl:modelReference="http://localhost/agensimo#GPSactive
                             http://localhost/agensimo#GPSposition"
      yasawsdl:serviceConcept="Precondition Result">
      <wsdl:input messageLabel="tns:getLocationGPSRequest"
        element="tns:getSignalGPS"
        sawsdl:modelReference="http://localhost/agensimo#GPSsignal"
        yasawsdl:serviceConcept="Input" />
      <wsdl:output messageLabel="tns:getLocationGPSResponse"
        element="tns:PositionGPS"
        sawsdl:modelReference="http://localhost/agensimo#GPScoordinates"
        yasawsdl:serviceConcept="Output" />
    </wsdl:operation>

    <wsdl:operation name="getApartmentBuyList"
    sawsdl:modelReference="http://localhost/agensimo#buy
                           http://localhost/agensimo#apart"
    yasawsdl:serviceConcept="Constraints Result">
      <wsdl:input messageLabel="tns:getApartmentBuyRequest"
        name="ApartmentBuyRequest" element="tns:ApartmentData"
        sawsdl:modelReference="http://localhost/agensimo#GPScoordinates"
        yasawsdl:serviceConcept="Input" />
      <wsdl:output messageLabel="tns:getApartmentBuyResponse"
        name="ApartmentBuyResponse" element="tns:Apartment"
        sawsdl:modelReference="http://localhost/agensimo#apart"
        yasawsdl:serviceConcept="Result" />
    </wsdl:operation>

    <wsdl:operation name="getMap"
      sawsdl:modelReference="http://localhost/agensimo#validApart"
      yasawsdl:serviceConcept="Precondition">
      <wsdl:input messageLabel="tns:getMapRequest"
        name="MapRequest" element="tns:MapData"
        sawsdl:modelReference="http://localhost/agensimo#APARTcoordinates"
        yasawsdl:serviceConcept="Input" />
      <wsdl:output messageLabel="tns:getMapResponse"
        name="MapResponse" element="tns:Map"
        sawsdl:modelReference="http://localhost/agensimo#MAPinformation"
        yasawsdl:serviceConcept="Result" />
    </wsdl:operation>
  </wsdl:interface>

  <wsdl:binding name="GeoLocApartBuyMapServiceSOAPBinding"
    interface="tns:GeoLocApartBuyMapServiceInterface" ...>    ...  </wsdl:binding>

  <wsdl:service name="GeoLocApartBuyMapWebService"
    interface="tns:GeoLocApartBuyMapServiceInterface">
    <wsdl:endpoint name="GeoLocApartBuyMapServiceEndpoint"
      binding="tns:GeoLocApartBuyMapServiceSOAPBinding"
      address="http://www.telecom-sudparis.eu/GeoLocApartBuyMapWebService/" />
  </wsdl:service>
</wsdl:description>
```

FIGURE 4.2 – Extrait d'une description YASA.

L'opération a comme précondition ("*Precondition*") le concept métier "*validApart*" : qui vérifie que l'appartement est disponible pour l'achat. L'input est les données de l'appartement, principalement sa localisation (lignes 53-56) et l'output est la carte avec les informations nécessaires pour se rendre à l'appartement, principalement l'image de carte et la redirection (lignes 57-60).

La Figure 4.2 illustre un extrait de description de service YASA écrite selon la récente spécification WSDL 2.0. La Figure 4.3 illustre uniquement la différence de l'annotation sémantique avec la spécification WSDL 1.1. La différence se résume à l'ajout d'un élément d'extension "*attrExtensions*" pour contenir les attributs de l'annotation sémantique. C'est la manière avec laquelle on étend l'élément "operation" en WSDL 1.1.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <wsdl:definitions
3    targetNamespace="http://www.telecom-sudparis.eu/GeoLocApartBuyMapWebService.wsdl11"
4  ...
5    name="GeoLocApartBuyMapWebService">
6
7    <wsdl:types>
8      <xs:import
9        namespace="http://www.telecom-sudparis.eu/GeoLocationMapWebService/schemas"
10       location="http://www.telecom-sudparis.eu/GeoLocationMapWebService.xsd" />
11   </wsdl:types>
12
13   <wsdl:message name="getLocationGPSRequest">
14     <wsdl:part name="getSignalGPS" type="xsd:string"></wsdl:part>
15   </wsdl:message>
16   <wsdl:message name="getLocationGPSResponse">
17     <wsdl:part name="PositionGPS" type="tns:Position"></wsdl:part>
18   </wsdl:message>
19 ...
20   <wsdl:portType name="GeoLocApartBuyMapServiceInterface"
21     sawsdl:modelReference="http://localhost/agensimo#terminalGPS
22                            http://localhost/agensimo#buy
23                            http://localhost/agensimo#apart"
24     yasawsdl:serviceConcept="Precondition Goal Effect">
25
26     <wsdl:operation name="getLocationGPS">
27   <sawsdl:attrExtensions
28       sawsdl:modelReference="http://localhost/agensimo#GPSactive
29                              http://localhost/agensimo#GPSposition"
30       yasawsdl:serviceConcept="Precondition Result"/>
31       <wsdl:input message="tns:getLocationGPSRequest"
32         element="tns:getSignalGPS"
33         sawsdl:modelReference="http://localhost/agensimo#GPSsignal"
34         yasawsdl:serviceConcept="Input" />
35       <wsdl:output message="tns:getLocationGPSResponse"
36         element="tns:PositionGPS"
37         sawsdl:modelReference="http://localhost/agensimo#GPScoordinates"
38         yasawsdl:serviceConcept="Output" />
39     </wsdl:operation>
40
41   <wsdl:binding name="GeoLocApartBuyMapWebServiceSOAP" ...>    ...  </wsdl:binding>
42
43   <wsdl:service name="GeoLocApartBuyMapWebService">
44     <wsdl:port binding="tns:GeoLocApartBuyMapWebServiceSOAP"
45       name="GeoLocApartBuyMapWebServicePortSOAP" />
46   </wsdl:service>
47 </wsdl:definitions>
```

FIGURE 4.3 – Exemple d'extrait d'une description YASA en spécification WSDL 1.1.

La section suivante présente des avantages de YASA par rapport à d'autres approches.

4.3.3 Avantages et contraintes autour de YASA

Je présente dans ce paragraphe les avantages de la contribution YASA principalement par rapport à OWL-S et WSMO. Premièrement, par rapport à WSMO et OWL-S, YASA offre de multiples avantages. Tout d'abord, les utilisateurs peuvent décrire, d'une manière incrémentale et compatible, à la fois à la sémantique et à la syntaxe des propriétés fonctionnelles et non fonctionnelles dans WSDL, qui est le langage le plus familier de la communauté des développeurs des services Web. Deuxièmement, en externalisant la sémantique des modèles de domaine (concepts métier) et les concepts techniques (fonctionnels, de qualité de service, de contexte, etc), on permet aux développeurs de services Web d'annoter leurs services Web avec une ontologie décrite par le langage de leur propre choix (comme UML ou OWL) contrairement à OWL-S ou WSMO qui exigent les leurs. En outre, OWL-S et WSMO définissent leurs propres ontologies de service contrairement à YASA, qui peut intégrer toute ontologie technique (de Contexte, de QoS ou une ontologie technique fonctionnelle de service). Par rapport aux travaux autour des correspondances entre OWL-S et SAWSDL présentés dans [68], YASA permet aussi à la communauté des développeurs de spécifier la correspondance entre les concepts de service (interfaces, operations, fault, etc.) et des concepts métier. Cette correspondance explicite peut être utilisée pour faire des correspondances entre les éléments des structures des descriptions YASA, des descriptions OWL-S, des descriptions WSMO, et de nombreux autres langages de description de services Web sémantiques. Par rapport au langage de description DIANE [44, 52], YASA se distingue par sa focalisation sur des normes à jour (SAWSDL) et des outils communs (des outils dédiés à WSDL et SAWSDL).

Les références des concepts dans les ontologies de services (de type ontologies techniques) doivent respecter une contrainte que je présente ici. En effet, pour un élément donné WSDL (service, interface, operation, etc.) on ne peut pas référencer n'importe quel concept dans une ontologie de services. Par exemple, un concept de comportement (qui peut être le concept "processus" dans une ontologie OWL service) ne peut être associé qu'aux éléments WSDL : "interface" et "operation". Un concept de comportement ne peut pas être associé à un "input". Par conséquent, une ontologie de services devrait être une extension du métamodèle de WSDL (des concepts de services Web WSDL comme "service", "interface", etc.) avec des concepts additionnels qui caractérisent (avec la relation hasProperty) les concepts WSDL. Avec cette condition, l'attribut "*serviceConcept*" de YASA d'un élément WSDL "E" ne devrait référencer que des concepts qui sont liés au concept "E" dans l'ontologie technique de services. Dans la section suivante, je présente une ontologie de service qui est une extension du métamodèle WSDL et qui contient d'autres concepts inspirés par les ontologies de services OWL-S et WSMO.

4.4 Ontologie TEchnique des Services Web (OTES)

YASA utilise une ontologie de services qui décrit les concepts techniques (*input, operation, precondition,* etc.) et les relations sémantiques entre ces concepts. L'objectif de cette section est de définir une ontologie technique de services pour le langage de description de service Web sémantique YASA. Cette ontologie intègre des concepts utiles du méta-modèle de WSDL et des ontologies dans OWL-S et WSMO. L'intégration de ces ontologies est réalisée avec différentes techniques d'appariement conformément à un processus d'alignement d'ontologies (mise en correspondance des concepts). L'ontologie résultante offre une couverture sémantique des concepts

spécifiques aux services Web. L'intérêt de cette mise en relation est l'obtention d'un modèle sémantique fédérateur des propriétés techniques des services Web qui servira à les annoter.

4.4.1 Besoin d'une ontologie technique

Dans cette section, je présente l'intérêt et l'avantage de la construction d'une ontologie technique de services Web afin d'être utilisée par l'approche de description YASA.

Dans OWL-S et WSMO, les aspects techniques des profils de services servent pour la découverte et les concepts techniques liés au comportement des services servent pour la composition. Mais OWL-S et WSMO, dans leurs approches d'ontologie de services, dépendent de langages de description d'ontologies spécifiques OWL et WSML. Ainsi, ils se limitent à un ensemble fermé de concepts bien définis.

Quant à SAWSDL, il permet d'utiliser tous types d'ontologies (OWL, WSML, UML, etc.) mais il ne définit pas explicitement une ontologie de services. Étant une extension de WSDL, SAWSDL ne nécessite pas beaucoup d'efforts pour les développeurs habitués à WSDL. L'expressivité du méta-modèle de WSDL présente une limite : elle ne permet pas d'intégrer des propriétés non fonctionnelles à la description. SAWSDL hérite de cette limite.

YASA propose d'annoter une description WSDL, non seulement par des concepts métier, mais aussi par des concepts de services définis dans une ontologie technique de service. Il reste à définir un modèle technique des concepts utilisés dans les services. L'idée est de s'inspirer des notions qui existent déjà et/ou de les enrichir afin de construire un modèle technique fonctionnel fédérateur autour des services Web. Pour ce faire, il faut pouvoir annoter sémantiquement l'aspect technique fonctionnel et fédérer les concepts techniques autour d'une ontologie de services Web.

Plus la description d'un service est expressive sur ses éléments fonctionnels, plus elle a de chance à être découverte par une requête qui partage avec elle des concepts similaires ou très proches. Afin d'améliorer la découverte, on a besoin de définir sous forme d'une ontologie, un ensemble de concepts et de relations entre eux, provenant du domaine des services et servant à mieux décrire les aspects fonctionnels. Ces concepts serviront à annoter la description de services YASA. Ainsi, cette description sera-t-elle enrichie par des références qui correspondent à des concepts techniques de services tels que la précondition, l'effet, l'opération, etc. Cette ontologie, à définir, serait une ontologie dite "technique". Grâce à cette ontologie, les services décrits en YASA pourront profiter de toutes les relations hiérarchiques et rhétoriques qui existent entre les concepts techniques pour annoter leurs éléments fonctionnels.

Approche de définition de l'ontologie de services

Pour définir une ontologie de services, je propose d'intégrer les méta-modèles et ontologies de services existants. En effet, la définition d'une "Ontologie TEchnique de Services" et de l'ensemble de ses concepts se base sur l'intégration des concepts techniques de services Web provenant des ontologies de OWL-S [69] et WSMO [58], du méta-modèle de WSDL [14] et de concepts retenus de différents travaux de recherche sur la description de services Web.

Je présente dans la suite comment nous avons rassemblé les concepts techniques dans une seule ontologie en nous basant sur la fusion et l'alignement de concepts provenant de plusieurs ontologies de services vues dans la littérature.

L'alignement de concepts est basé principalement sur l'identification de correspondances entre les concepts. Les correspondances sont les relations entre les éléments de deux modèles sémantiques (par exemple, les ontologies). La correspondance entre deux concepts indique une similarité selon une mesure donnée [41]. Le "matching" ou appariement est un ensemble de mécanismes permettant de spécifier des correspondances entre les concepts [30]. Ceci peut être réalisé par une ou plusieurs méthodes de comparaison dites "matchers" ou apparieurs.

Outil d'alignement pour la construction de l'ontologie de services

Pour intégrer les ontologies de services, je me base sur le système d'appariement OMIE conçu au sein de notre équipe de recherche et qui est une des contributions de la thèse de Abdeltif Elbyed [20].

Afin de produire un alignement entre deux ontologies, OMIE exécute un processus qui contient plusieurs étapes :

1. Calcul des similarités terminologiques (syntaxiques et linguistiques)
2. Agrégation des valeurs de similarité,
3. Filtrage des correspondances (selon les relations structurales et sémantiques)
4. Validation de l'alignement.

4.4.2 Construction de l'ontologie technique de services Web OTES

Principe de la construction de l'ontologie OTES

Dans la suite, je présente l'application d'une adaptation du processus d'alignement présenté dans la thèse de Abdeltif Elbyed [20] à la construction de l'ontologie technique de services Web. Voici les étapes que j'ai identifié pour cette construction :

1. Identification des concepts des différentes ontologies à apparier,
2. Calcul des similarités terminologiques et agrégation des valeurs,
3. Filtrage et validation des correspondances.

Identification des concepts à apparier

Pour construire l'ontologie qui intègre les ontologies OWL-S, WSMO, le méta-modèle WSDL et d'autres modèles, je effectue l'alignement de ces différentes ontologies. Je commence par identifier les concepts à apparier et provenant des différents modèles. Dans la suite, j'applique manuellement une à une les techniques de l'approche OMIE. La Figure 4.4 liste les principaux concepts à apparier.

OWL-S	WSMO	WSDL	Autres
Effect	Choreography	Description	Comment
ServiceModel	Orchestration	InterfaceFault	Category
Precondition	Capability	Interface	Community
Process	Goal	InterfaceOperation	Version
AtomicProcess	Interface	Operation	Status
SimpleProcess	Effect	Service	Business
CompositeProcess	Assumption	BindingOperation	Actor
Parameter	Service	Endpoint	Provider
Profile	Postcondition	Input	Constraint
Local	Precondition	BindingFault	Capacity
Participant	WebService	Binding	Port
Input		Documentation	
Output		Output	
Result			
Service			

FIGURE 4.4 – Principaux concepts à apparier.

Calcul des similarités terminologiques

Soit la sémantique de symboles suivante : le symbole " !" désigne une non similarité élevée (concepts sans relations), et le symbole "#" désigne que les deux concepts proviennent d'une même ontologie et ne doivent pas être alignés. La Figure 4.5 présente les résultats d'appariement entre quelques concepts de l'exemple du tableau n'indique que les valeurs de distance inférieures à la somme des longueurs de deux termes sinon il affiche " !". Dans La Figure 4.5, par exemple :

- 1ère ligne, 1ère colonne, la similarité entre Input (WSDL) et Input (OWL-S) est égal à 0,
- 1ère ligne, 5ème colonne, la similarité entre Input (WSDL) et Output (OWL-S) est égal à 5, car il faut retirer les deux lettres "in" puis ajouter les trois lettres "out".
- 2ème ligne, 2ème colonne, la similarité entre Interface (WSDL) et Input (WSDL) est égal à # car les deux proviennent du même méta modèle.

	$Input_{OWLS}$	$Service_{WSDL}$	$Interface_{WSMO}$	Capacity	$Output_{OWLS}$	Result	Precondition	...
$Input_{WSDL}$	0	!	!	!	5	7	!	...
$Interface_{WSDL}$	8	#	0	!	!	!	!	...
Precondition	#	!	!	!	#	#	0	...
Postcondition	!	!	#	!	!	!	5	...
InterfaceFault	!	#	5	!	!	!	!	...
WebService	!	3	#	!	!	!	#	...
Capability	!	!	#	4	!	12	#	...
$Output_{WSDL}$	5	#	!	!	0	8	!	...
...	...	...	...	...	...	...	...	...

FIGURE 4.5 – Extrait des résultats après application de l'appariement terminologique.

L'apparieur utilisé pour calculer les similarités se base sur la distance de Levenshtein. La similarité entre deux chaînes de caractères est le nombre minimal d'insertions et de suppressions de caractères requis pour transformer une chaîne en une autre. La Figure 4.6 présente un extrait des valeurs de similarités résultant après application . J'adopte ici la définition d'apparieur à base du dictionnaire WordNet : je me base sur les liens sémantiques entre les termes en langage naturel pour calculer la valeur de similarité SV : les synonymes sont des entités équivalentes (SV=1), les hyponymes sont des entités recouvrantes (SV=0.5) et sinon il n'y a pas de lien (SV=0).

	$Input_{OWLS}$	$Service_{WSDL}$	$Interface_{WSMO}$	Capacity	$Output_{OWLS}$	Result	Precondition	...
$Input_{WSDL}$	1	0	0	0	0.7	0.5	0	...
$Interface_{WSDL}$	0.25	0	1	0	0	0	0	...
Precondition	0	0	0	0	0	0	1	...
Postcondition	0	0	0	0	0	0	0.7	...
InterfaceFault	0	0	0.7	0	0	0	0	...
WebService	0	0.7	0	0	0	0	0	...
Capability	0	0	0	0.7	0	0.1	0	...
$Output_{WSDL}$	0.7	0	0	0	1	0.5	0	...
...	...	...	...	...	...	...	...	...

FIGURE 4.6 – Extrait des résultats après application des règles de la valeur de similarité.

	$Input_{OWLS}$	$Service_{WSDL}$	$Interface_{WSMO}$	Capacity	$Output_{OWLS}$	Result	Precondition	...
$Input_{WSDL}$	1	0	0	0	0	0	0	...
$Interface_{WSDL}$	0	0	1	0	0	0	0	...
Precondition	0	0	0	0	0	0	1	...
Postcondition	0	0	0	0	0	0	0	...
InterfaceFault	0	0	0	0	0	0	0	...
WebService	0	0.5	0	0	0	0	0	...
Capability	0	0	0	1	0	0	0	...
$Output_{WSDL}$	0	0	0	0	1	0	0	...

FIGURE 4.7 – Extrait des résultats après application d'appariement linguistique.

Après un calcul de similarités syntaxiques avec la distance de Levenshtein, j'applique un calcul de similarités linguistiques grâce à l'appariseur à base du dictionnaire WordNet : je me base sur les liens sémantiques entre les termes en langage naturel pour calculer la valeur de similarité : les synonymes sont des concepts équivalents, ce qui vaut la valeur 1, les hyponymes sont des concepts recouvrants, ce qui vaut la valeur 0.5, et s'il n'y a pas de lien alors c'est la valeur 0. La Figure 4.7 présente un extrait des valeurs de similarités résultant après application d'appariement linguistique. Après les calculs de similarités, j'utilise la formule de combinaison parallèle des valeurs de similarités comme elle est présentée dans [20]. Les valeurs sont combinées afin de calculer une seule valeur par couple de concepts. La Figure 4.8 présente un extrait des résultats de la combinaison des valeurs de similarités.

	$Input_{OWLS}$	$Service_{WSDL}$	$Interface_{WSMO}$	Capacity	$Output_{OWLS}$	Result	Precondition	...
$Input_{WSDL}$	1	0	0	0	0.2	0.1	0	...
$Interface_{WSDL}$	0.25	0	1	0	0	0	0	...
Precondition	0.08	0	0	0	0	0	1	...
Postcondition	0	0	0	0	0	0	0.2	...
InterfaceFault	0	0	0.2	0	0	0	0	...
WebService	0	0.56	0	0	0	0	0	...
Capability	0	0	0	0.9	0	0.03	0	...
$Output_{WSDL}$	0.2	0	0	0	1	0.1	0	...

FIGURE 4.8 – Extrait des résultats de la combinaison des valeurs de similarités.

Filtrage et validation des correspondances

A ce stade, toutes les valeurs obtenues ne sont que des hypothèses d'alignement. Toutes ces hypothèses doivent être filtrées puis validées avant d'établir l'alignement de tous les concepts. A partir du tableau de la Figure 4.8, on les note comme suit :

Hp<≡,Input,Input,3,1>	Hp<≡,Postcondition,Postcondition,3,0.2>
Hp<≡,Input,Result,3,0.1>	Hp<≡,InterfaceFault,Interface,3,0.2>
Hp<≡,Input,Output,3,0.2>	Hp<≡,WebService,Service,3,0.56>
Hp<≡,Interface,Input,3,0.25>	Hp<≡,Capability,Capacity,3,0.9>
Hp<≡,Interface,Interface,3,1>	Hp<≡, Capability,Result,3,0.03>
Hp<≡,Precondition,Input,3,0.08>	Hp<≡, Output,Result,3,0.1>
Hp<≡,Precondition,Precondition,3,1>	Hp<≡, Output,Output,3,1> . . .

Chacune de ces hypothèses d'alignement (Hp) contient un cinq-tuplet :

Hp <R, c, c', ConfHp, SVHp> avec :

- Hp : L'hypothèse d'alignement,
- R : Relation identifiée par les appariеurs entre le couple de concepts c et c',
- c : Un premier concept à comparer,
- c' : Un deuxième concept auquel on compare c,
- ConfHp : le degré de confiance de l'hypothèse d'alignement
- SVHp : la valeur de similarité produite par la combinaison des valeurs de similarités

Par exemple, la valeur de similarité finale entre les concept Input (WSDL) et Input (OWL-S) est 1 (ligne 1, colonne 1), ce qui vaut l'équivalence, alors que la valeur de similarité finale entre Input et Output est de 0.2 (ligne 2, colonne 1), ce qui est une valeur très faible pour être considérée comme une potentielle correspondance dans l'alignement. On retrouve aussi des concepts proches mais pas équivalents comme Capability et Capacity avec une valeur de similarité de 0.9 (ligne 4, colonne 2).

A ce stade, je continue à appliquer la procédure d'alignement comme il est indiqué dans [20]. J'applique des raffinements des valeurs de similarités grâce à un appariement sémantique puis un appariement structural.

Finalement, j'applique l'ensemble des filtres nécessaires cités dans [20] sur les hypothèses (filtres structuraux, sémantiques et à base de seuil). Par exemple, dans un filtre à base de seuil, je trie en ordre décroissant les hypothèses selon la valeur de leurs valeurs de similarités et j'élimine les hypothèses d'alignement les moins pertinentes. A ce stade, il ne reste qu'à valider l'alignement et exporter l'ontologie résultat. La Figure 4.9 présente une liste des principaux concepts de l'ontologie technique après validation des alignements. Ce sont les concepts ayant des hypothèses d'alignement avec de grandes valeurs de similarité. En effet de chaque couple de concepts d'une hypothèse validée, nous gardons un seul nom qui représente dorénavant les deux concepts provenant de deux ontologies ou méta modèles distincts, par exemple : *Effect, Service, Category,* etc. La Figure 4.10 illustre quelques principaux concepts de l'ontologie OTES avec leurs relations hiérarchiques. On découvre des relations telles que "is-a" qui relie Input et Output à Parameter, ou qui relie aussi Category et Version à Classification. La Figure 4.11 illustre quelques principaux concepts de l'ontologie OTES avec leurs relations rhétoriques. Par exemple, la relation *inCondition* relie les concepts *Result* à *Précondition*. Je n'ai pu insérer tout l'arbre de l'ontologie technique de services vu la taille de la figure.

Effect	Service	Endpoint
ServiceModel	Choreography	BindingFault
Precondition	Orchestration	Binding
Process	Capability	Documentation
AtomicProcess	Goal	Comment
SimpleProcess	Interface	Category
CompositeProcess	Effect	Community
Parameter	Assumption	Version
Profile	Postcondition	Status
Local	Description	Business
Participant	InterfaceFault	Actor
Input	InterfaceOperation	Provider
Output	Operation	Constraint
Result	BindingOperation	Port

FIGURE 4.9 – Liste des principaux concepts de l'ontologie technique.

FIGURE 4.10 – Principaux concepts de l'ontologie OTES avec leurs relations hiérarchiques.

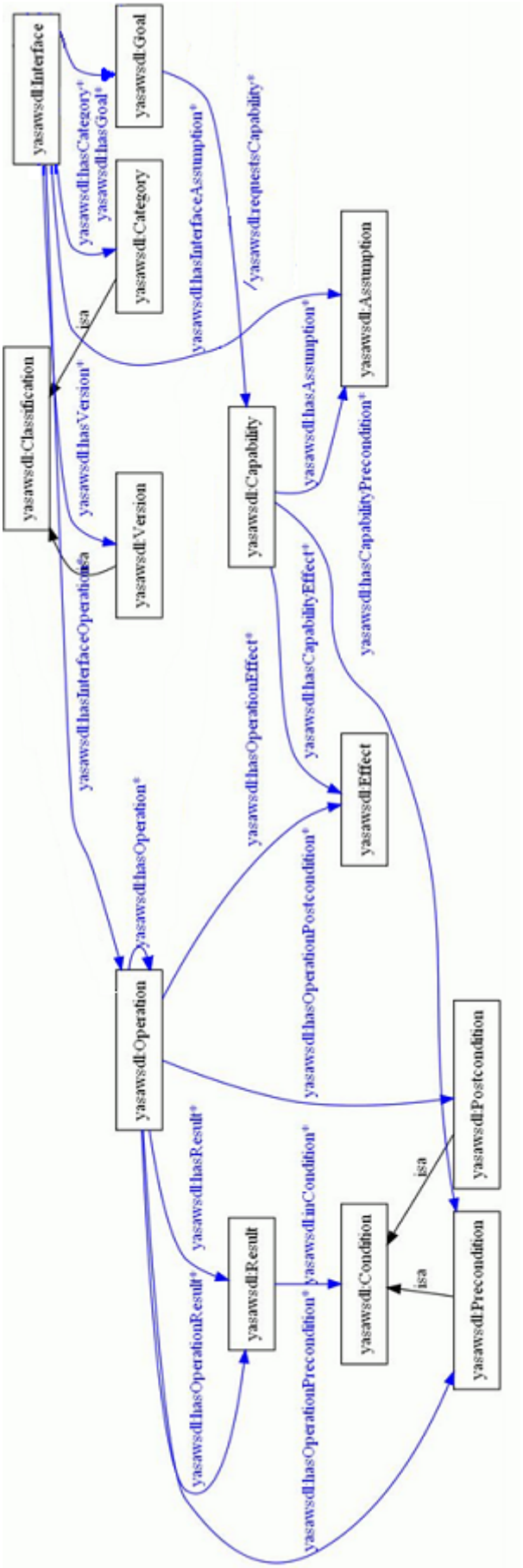


FIGURE 4.11 – Principaux concepts de l'ontologie OTES avec leurs relations rhétoriques.

4.5 Conclusion

Dans ce chapitre, j'ai présenté ma contribution en termes de description de services Web sémantiques. Dans une première partie, j'ai présenté le langage YASA et comment il intègre la sémantique en se basant sur l'utilisation de deux types d'ontologies : "ontologie technique" et "ontologie de domaine". Ce principe a permis à YASA de gagner en expressivité par rapport à WSDL, SAWSDL, OWL-S et WSMO puisque YASA s'inspire des avantages de leurs approches et comble leurs lacunes.

La contribution de ce chapitre vise une approche de description à la fois fédératrice et basée sur les standards. Pour cela, j'ai construit une ontologie technique de service par intégration des ontologies OWL-S, WSMO et le méta-modèle WSDL ainsi que d'autres concepts inspirés de mes lectures. La seconde partie de ce chapitre a présenté l'ontologie technique des services Web sémantiques OTES, utilisée par l'approche YASA. Le chapitre suivant présente les mécanismes d'appariement sémantique entre des descriptions YASA utilisées pour améliorer la découverte sémantique des services Web.

Chapitre 5

L'appariement sémantique pour la découverte de services

Sommaire

5.1 Introduction

Pour réussir une découverte sémantique, il faut proposer un algorithme d'appariement pertinent entre une requête de service et les descriptions de services publiés. Les nombreux services disponibles aujourd'hui sur des annuaires purement syntaxiques, voire même sur certains annuaires supportant la sémantique (e.g. UDDI, ebXML) continuent d'exposer des problèmes de pertinence et d'ambiguïté par rapport à l'appariement sémantique. Ceci influe négativement sur le coût d'une découverte. D'une part, les mécanismes de découverte de services Web traditionnels, consistent uniquement à apparier syntaxiquement les termes d'une requête à une description offerte. D'autre part, les mécanismes sémantiques actuels associent des concepts à une description sans préciser la nature sémantique de ces annotations.

Pour comparer la description d'un service requis à celle d'un service offert, le processus d'appariement sémantique effectue un appariement à chaque niveau d'éléments des descriptions (appariement entre interfaces requises et offertes, appariement entre opérations requises et offertes...). Ceci est appelé un appariement élémentaire. L'ensemble des résultats de tous les

appariements élémentaires peut être agrégé selon différents principes afin de calculer le degré global d'appariement sémantique.

Dans ce cadre, mes travaux ont conduits à une nouvelle approche d'appariement élémentaire et à trois principales approches d'agrégation pour l'appariement global [11] : Min-Average, Cupid et Combinatory.

Dans ce chapitre, je présente les bases de l'appariement sémantique élémentaire (Section 5.2), les principes des approches d'agrégation sémantique pour l'appariement global : l'agrégation sémantique d'appariement global Min-Average (Section 5.3), l'agrégation Cupid et l'agrégation Combinatory (Section 5.4). Cette approche d'appariement sémantique m'a permis de concevoir et réaliser un nouveau matchmaker sémantique (apparieur sémantique) pour les services Web. Ce matchmaker sémantique est adapté aux services Web décrits en standard *WSDL*, en *SAWSDL* et en *YASA* [11].

5.2 Appariement sémantique élémentaire

Pour réaliser un appariement sémantique entre deux descriptions, je propose une approche d'appariement global basé sur l'agrégation des appariements sémantiques élémentaires entre les éléments composants de chacune des descriptions de services. Dans ce qui suit, je présente le principe de l'appariement global ainsi que mon approche d'appariement sémantique élémentaire.

5.2.1 Principe de l'appariement global

L'appariement est dit élémentaire lorsqu'il s'applique à la comparaison de couples d'éléments de même nature de part et d'autre : interface offerte avec interface requise, opération offerte avec opération requise, inputs offerts avec inputs requis et outputs offerts avec outputs requis. Un degré d'appariement élémentaire représente donc à quel point deux éléments WSDL sont similaires.

L'appariement est dit global lorsqu'il produit l'évaluation finale de toutes les valeurs d'appariement élémentaires, une agrégation qui permet de consolider toutes les valeurs propres aux différents éléments d'un service et de les substituer par une seule valeur, propre au service, appelée degré d'appariement.

La phase d'appariement sémantique est importante dans la découverte de services Web sémantiques. Elle se déroule sur deux étapes :

1. Calcul des valeurs d'appariements élémentaires entre éléments *WSDL* des descriptions de services,
2. Agrégation des valeurs d'appariements élémentaires et calcul des degrés globaux d'appariement.

Dans ce qui suit, je présente une échelle de degrés d'appariement sémantique ainsi que les principes de l'appariement sémantique élémentaire.

5.2.2 Degrés d'appariement sémantique élémentaire

L'appariement élémentaire repose sur une approche logique et déductive [11]. Les éléments de la partie fonctionnelle d'une description *YASA* sont annotés sémantiquement avec des concepts logiques provenant d'ontologies. L'appariement de ces éléments se résume à un appariement entre les concepts qui les annotent. La similarité entre deux concepts est évaluée par un degré d'appariement. Les degrés d'appariement élémentaire utilisés dans mon approche sémantique sont :

- "Exact" : si le concept offert et le concept requis sont les mêmes (ou équivalents),
- "Subsumes" : si le concept offert est un sous concept du concept requis,
- "Subsumes-by" : si le concept requis est un sous concept du concept offert,
- "Has-same-class" : si les concepts (offert et requis) sont sous concepts du même concept,
- "Unclassified" : si au moins un des deux concepts n'est pas spécifié,
- "Fail" : si les deux concepts sont spécifiés mais aucune relation ne peut être déterminée entre eux.

Ces degrés d'appariement sont fortement inspirés de la théorie de subsomption utilisée pour déterminer les relations logiques et donc les valeurs d'appariement entre deux concepts. De plus, certains degrés permet de distinguer avec précision certains cas d'appariement qui manquaient dans les travaux précédents [78, 53, 45, 46] et [36]. Certains degrés sont propres à mon approche (Has-Same-Class et Unclassified) et les autres trouvent leurs origines dans l'état de l'art sur l'appariement des services.

5.2.3 Principes de l'appariement sémantique élémentaire

Une des principales motivations de la proposition d'une nouvelle approche d'appariement élémentaire est l'amélioration de la précision. Celle-ci est réalisée grâce à la prise en compte non uniquement de l'information sémantique élémentaire mais en plus de sa nature. "Traditionnellement", un élément d'une description *WSDL* peut être simplement associé à un ou plusieurs concepts de domaine (e.g. dans *SAWSDL* et *OWL-S*). Des difficultés voire des erreurs peuvent être induites au moment de l'appariement. Une annotation avec un même concept peut avoir deux significations sémantiques différentes. Un appariement entre un élément requis et un élément offert d'un service peut être erroné si un même concept les annotant n'a pas la même signification. Prenons un premier exemple, pour une réservation dans un hôtel, l'opération de paiement peut être annotée par le concept de carte bancaire. Mais dans un premier service, le solde pourrait être retiré dès le départ, alors que dans un autre service, le solde n'est retiré qu'après la fin du séjour. Prenons un deuxième exemple, un appariement entre deux éléments chacun annotés par plusieurs concepts. En effet, chacun d'eux pourrait porter, à l'élément *WSDL* décrit, une signification sémantique différente (e.g. une opération de retrait en DAB doit à la fois, satisfaire une précondition telle que "carte bancaire valide" et avoir comme effet le "débit du solde"). En conclusion, plusieurs annotations peuvent annoter plusieurs aspects techniques d'un élément *WSDL*. Il est clair que dans tous les cas semblables à ces deux exemples, l'appariement élémentaire, l'appariement global, et par suite la découverte risquent tous d'être erronés. Pour éviter toute erreur et ambiguïté, mon approche de découverte remonte à l'origine de la problématique, à savoir : enrichir l'annotation sémantique par un aspect technique de description.

Je rappelle ici le principe de l'annotation. Chaque élément est annoté par une liste de concepts de domaines qui correspondent à une liste de concepts dits "techniques". Chaque concept de

la première liste correspond au concept du même rang dans la deuxième liste. Ces concepts techniques proviennent d'une ontologie que je propose et qui est inspirée et enrichie par des concepts provenant de méta-modèles et d'ontologies de services Web de l'état de l'art. Cette ontologie offre la plupart des concepts techniques que les fournisseurs de services Web ainsi que les utilisateurs peuvent associer à un élément *WSDL* afin de préciser la nature de son annotation sémantique (e.g. précondition ou postcondition pour l'exécution d'une opération, effet d'un output, résultat associé à une interface...).

Dans mon approche, je peux associer un élément (e.g. opération "acheter") à un premier concept dit "technique" (e.g. précondition), lui même associé à un deuxième concept dit "de domaine" (e.g. réservationValide). Aussi, je peux associer un élément à une liste de concepts techniques eux même associés dans l'ordre à une liste de concepts de domaines : le premier concept technique est associé au premier concept de domaine, le deuxième concept technique au deuxième concept de domaine, ainsi de suite. L'appariement élémentaire se résume par les appariements entre chacune des paires (concept technique, concept de domaine) requises et celles offertes.

Pour assurer une meilleure pertinence avec une telle approche, j'adopte le principe suivant :

- Le degré d'appariement de deux concepts de domaine comparés est l'un des degrés : Exact, Subsumes, Subsumes-by, Has-same-class, Unclassified et Fail.
- Si le concept technique offert et le concept technique requis sont les mêmes (ou équivalents), les deux concepts de domaine associés seront comparés et leur degré d'appariement sera le degré d'appariement élémentaire final local à agréger,
- Si le concept technique offert et le concept technique requis ne sont pas les mêmes (ou ne sont pas équivalents), on les apparie et leur degré d'appariement sera l'un des degrés : Subsumes, Subsumes-by, Has-same-class, Unclassified et Fail.
  - Si c'est Fail alors les concepts de domaine ne seront pas comparés, et on retourne Fail comme degré d'appariement élémentaire final local à agréger,
  - Sinon le degré d'appariement élémentaire final local à agréger sera une pondération du degré d'appariement des concepts de domaine par le degré d'appariement des concepts techniques.

La Figure 5.1 explique le mécanisme d'un appariement élémentaire. C'est le même mécanisme pour un appariement de deux interfaces, de deux opérations, de deux inputs ou de deux outputs. Au niveau de la description *WSDL*, les concepts techniques sont spécifiés à l'intérieur d'un élément grâce à l'attribut "serviceConcept" et les concepts de domaine sont spécifiés grâce à l'attribut "modelReference". L'attribut "serviceConcept" spécifie des concepts provenant de l'ontologie technique des services Web et l'attribut "modelReference" spécifie des concepts provenant de diverses ontologies de domaine. Le mécanisme d'appariement élémentaire se compose de deux étapes successives pour chaque élément WSDL annoté. La première étape consiste à comparer un par un les concepts techniques (voir la partie gauche de la Figure 5.1) et après chaque comparaison, le mécanisme réalise la deuxième étape qui consiste à comparer les concepts de domaine correspondant (voir la partie droite de la Figure 5.1).

Puisque mon objectif n'est pas l'appariement de concepts de deux ontologies différentes, alors je suppose que chaque fois que deux concepts sont comparés, ils doivent simplement appartenir à la même ontologie, mais rien n'empêche que deux autres concepts à comparer proviennent d'une autre ontologie. Pour moi, il est très important que mon approche permette l'annotation par plusieurs ontologies de domaine, ceci offre plus de flexibilité et de richesse à la description sémantique.

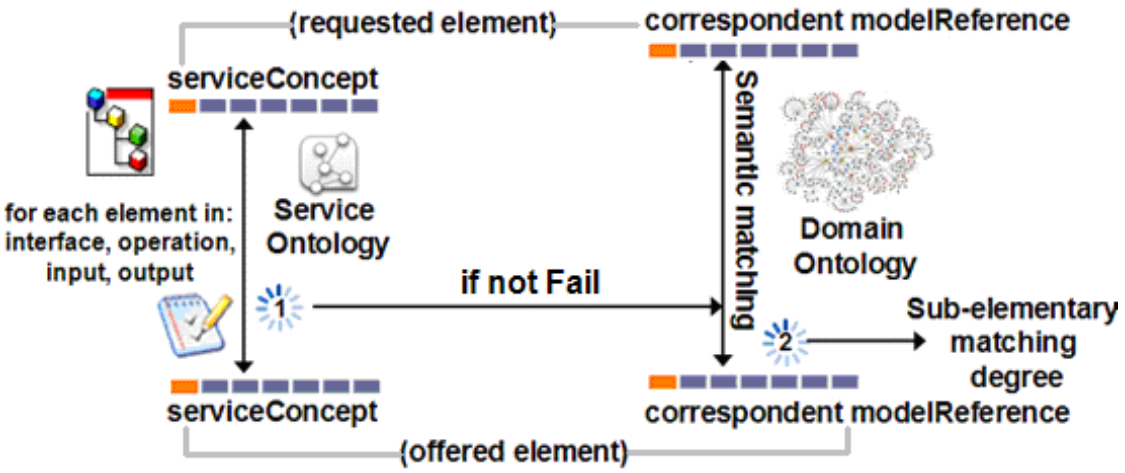


FIGURE 5.1 – Appariement élémentaire sémantique.

Dans mon approche, les degrés d'appariement élémentaire à chaque niveau (*interface, operation, input* ou *output*) peuvent être agrégés de trois façons. Les trois approches d'agrégation sont présentées dans la section suivante. J'ai défini une première approche basée sur mon principe de l'appariement Min-Average [11] et proposé deux autres algorithmes inspirés des approches de comparaisons de graphes Cupid [65] et Combinatory [33].

Pour l'agrégation des valeurs d'appariement élémentaire obtenues et le calcul du degré d'appariement global de deux descriptions YASA, je présente dans la suite les trois approches d'agrégation suivantes : l'approche Min-Average (Section 5.3), l'approche basée sur Cupid (Section 5.4) et l'approche basée sur Combinatory (Section 5.4).

5.3 Agrégation sémantique : Min-Average

5.3.1 Principes de fonctionnement

Pour l'agrégation des degrés d'appariement élémentaires en appliquant la méthode de l'appariement Min-Average, il faut calculer pour chaque élément annoté de la description *WSDL* offerte sa valeur d'appariement élémentaire avec l'élément correspondant dans la requête [11]. L'ordre de calcul entre ces éléments est connu d'avance : les interfaces, puis les opérations, ensuite les *inputs* et enfin les *outputs*. Cet ordre n'a aucun impact sur le calcul. Le plus important est que toutes les valeurs d'appariements doivent être obtenues afin de calculer le degré d'appariement global. L'algorithme de calcul du degré d'appariement global appliquant l'approche d'agrégation Min-Average est détaillé dans la Figure 5.2.

Dans l'algorithme, les termes ***rqt*** et ***adv*** dénotent respectivement les termes "requis" et "offert", ainsi, les variables *rqtInterface* et *advInterface* représentent respectivement l'interface de la description requise et l'interface de la description offerte. De même, les variables *rqtOperation*, *rqtInputs* et *rqtOutputs* représentent respectivement les éléments *operation*, *input*s et *output*s de la description requise (la requête) alors que les variables *advOperation*, *advInputs* et *advOutputs* représentent respectivement les éléments *operation*, *input*s et *output*s de la description offerte (l'offre).

```
1    InterfaceMD = SemanticMatching (rqtInterface , advInterface) ;
2    ListMD = ListMD :: InterfaceMD ;
3
4    if ( InterfaceMD > a ) {
5       loop {
6              OperationMD = SemanticMatching (rqtOperation , advOperation ) ;
7              ListMD = ListMD :: OperationMD ;
8              if ( OperationMD > b )
9                 loop {
10                       InputsMD = SemanticMatching ( rqtInputs , advInputs ) ;
11                       OutputsMD = SemanticMatching ( rqtOutputs , advOutputs ) ;
12                       ListMD = ListMD :: InputsMD ;
13                       ListMD = ListMD :: OutputsMD ;
14                } end loop
15             GlobalOperationMD = ( ( InputsMD + OutputsMD + OperationMD ) / 3 ) ;
16             TotalGlobalOperations = OperationsMD + GlobalOperationMD
17       }end loop
18
19       OperationsMD = ( GlobalOperationMD / nbrRequestedOperations ) ;
20
21       Min = Min ( ListMD ) ;
22       Average = ( InterfaceMD + OperationsMD ) / 2 ;
23       YasaMD = ( Min , Average ) ;
24       return YasaMD ;
25   }
26   else
27       return 0
```

FIGURE 5.2 – Algorithme d'appariement élémentaire et agrégation avec l'approche Min-Average.

Le terme **MD** signifie Degré d'appariement (*Matching Degree*) et :

- *InterfaceMD* dénote le degré d'appariement élémentaire entre une interface offerte et une interface requise (ligne 1),
- *OperationMD* dénote le degré d'appariement élémentaire entre une opération offerte et une opération requise (ligne 6),
- *InputsMD* dénote le degré d'appariement élémentaire entre des inputs offerts et des inputs requis (ligne 10),
- *OutputsMD* dénote le degré d'appariement élémentaire entre des outputs offerts et des outputs requis (ligne 11).
- *ListMD* dénote la liste de tous les degrés d'appariement élémentaire aux niveaux d'inputs, d'outputs, d'opérations et d'interface (lignes 2,7,12,13 et 21).

L'appariement Min-Average commence par agréger le degré d'appariement de l'élément englobant opération ainsi que ses sous éléments input et output, c'est le degré d'appariement global d'une opération *GlobalOperationMD* (ligne 15). La somme des degrés d'appariement globaux de toutes les opérations est notée *TotalGlobalOperations* (ligne 16). Cette somme est divisée par le nombre total des opérations requises *nbrRequestedOperations* pour avoir un degré d'appariement global final au niveau des opérations, noté *OperationsMD* (ligne 19).

Pour le calcul du degré d'appariement global des deux descriptions de services *YasaMD*, il faut :

- identifier la valeur minimale *Min* (ligne 21) parmi les degrés d'appariement élémentaire,
- calculer la moyenne *Average* (ligne 22) entre le degré d'appariement des opérations *OperationsMD* et le degré d'appariement des interfaces *InterfacesMD* des descriptions.
- retourner le degré d'appariement sémantique final : c'est le couple de valeurs (Min, Average) (ligne 23) .

La variable "a" (ligne 4) présente le seuil d'appariement entre deux interfaces pour décider de l'intérêt des appariements élémentaires entre leurs opérations. La variable "b" (ligne 8) représente le seuil d'appariement entre deux opérations pour décider de l'intérêt des appariements élémentaires entre leurs inputs et leurs outputs. En définissant ces seuils, la découverte est devenue plus sélective et plus rapide. Elle est plus sélective car seuls les degrés d'appariement les plus élevés sont sélectionnés. Ainsi, elle est plus rapide puisqu'il n'y aurait pas de perte de temps en essayant d'apparier les inputs et les outputs provenant d'opérations requises et offertes qui, eux-mêmes, ne s'apparient pas (leur appariement subit un échec).

5.3.2 Discrétisation des degrés d'appariement élémentaire

Dans ce qui suit, j'explique comment associer aux degrés d'appariement élémentaire des valeurs entières et comment les utiliser pour calculer le degré global d'appariement.

TABLE 5.1 – Les valeurs entières des degrés d'appariement élémentaires

Degré d'appariement élémentaire	Interface / Operation / Output	Input
EXACT	5	5
SUBSUMES	4	3
SUBSUMED BY	3	4
HAS SAME CLASS	2	2
UNCLASSIFIED	1	1
FAIL	0	0

Je propose une échelle de valeurs pour les degrés d'appariement allant de 0 à 5, avec :

– 0 : la valeur affectée en cas d'échec d'appariement,
– 5 : la valeur affectée en cas d'appariement exact ou équivalence.

Pour les interfaces, les opérations et les outputs, plus l'appariement est précis, plus la valeur est grande (voir Table 5.1, colonne 1). Les inputs présentent un cas spécifique que je détaille ici. En effet, les offres de services ayant des concepts d'inputs plus généraux que ceux des inputs du service requis, vont satisfaire toute requête d'utilisateur puisque ils fournissent des inputs toujours équivalents à ce qui est requis. Mais les offres de services ayant des concepts d'inputs plus spécifiques que ceux des inputs du service requis, peuvent parfois ne pas satisfaire l'utilisateur. Pour cela, il est important de prendre en compte ce détail, ainsi je propose pour les inputs de substituer le degré "SUBSUMES" par la valeur 3 et le degré "SUBSUMED BY" par la valeur 4. Les valeurs des degrés ne sont pas arbitraires. Chacune d'entre elles est proportionnelle à la similarité, plus les éléments appariés sont similaires, plus la valeur augmente. Le cas particulier de l'input prend en compte cette règle (voir Table 5.1, colonne 2).

5.3.3 Degré d'appariement final YASA : étude de cas

Bien définir les degrés d'appariement peut améliorer la précision mais ne résout pas tous les éventuels problèmes de l'appariement. Pour illustrer encore une fois l'intérêt de mon approche, je discute dans la suite d'un cas d'appariement face auquel nous réalisons une meilleure découverte, là où d'autres approches présentent des lacunes, déjà discutées dans la section 3.5.

Le cas étudié prévoit l'appariement d'une requête à deux services donnés. Les services se ressemblent beaucoup, ils ont les mêmes noms et nombres d'éléments mais leurs annotations sémantiques diffèrent, ce qui est un cas très fréquent. Dans la Table 5.2, chaque ligne présente les degrés des appariements élémentaires entre les éléments d'un service offert et les éléments du service requis, on a :

- une interface Interface-I,
- une première opération Operation1 avec un input Input1 et un output Output1,
- une deuxième opération Operation2 sans inputs ni outputs.

Je propose d'évaluer maintenant l'agrégation des degrés d'appariement de deux approches pour calculer le degré d'appariement global. Dans la Table 5.2, l'avant dernière colonne présente le résultat obtenu en appliquant le principe d'agrégation "Valeur minimale", un principe connu et utilisé dans des travaux tel que SAWSDL-MX pour apparier les descriptions SAWSDL. Comme son nom l'indique, il sélectionne comme degré d'appariement final la valeur minimale parmi toutes les valeurs d'appariement élémentaire. La dernière colonne présente le résultat obtenu en appliquant le principe d'agrégation Average (la moyenne). Ici, j'étudie le cas particulier de deux services. Le premier service offre deux opérations qui s'apparient exactement avec les deux opérations de la requête (chacune a 5 comme degré d'appariement). Le deuxième service offre deux opérations avec deux concepts plus spécifiques que ceux des deux opérations de la requête (chacune a 3 comme degré d'appariement).

TABLE 5.2 – Exemples d'agrégations Valeur minimale et moyenne.

Services	Interface-I	Operation2	Operation1	Input1	Output1	Valeur Minimale	Valeur moyenne
1er Service	5	5	5	3	5	3	4.83
2ème Service	5	3	3	5	5	3	4.33

Pour évaluer l'efficacité de l'approche "Valeur minimale", j'applique le principe de l'agrégation sur les degrés d'appariement élémentaire des deux services et je compare les résultats.

- Dans la ligne 1 : l'agrégation sélectionne parmi { 5,5,5,3,5 }, le plus petit des degrés d'appariement à agréger, c'est-à-dire 3,
- Dans la ligne 2 : l'agrégation sélectionne parmi { 5,3,3,5,5 }, le plus petit des degrés d'appariement à agréger, c'est-à-dire 3,

Donc, finalement les deux services publiés sont considérés à égalité si on ne se fie qu'aux résultats finaux. Mais il est clair que le premier service satisfait beaucoup mieux la requête d'utilisateur que le deuxième service. En effet, le premier service affiche équivalent sur tous les éléments sauf pour l'input qui affiche le deuxième meilleur degré (sur l'échelle des degrés des inputs). D'autre part, le deuxième service risque de ne pas assez satisfaire la requête avec deux opérations ayant le degré moyen "SUBSUMED BY". La différence des degrés d'appariement au niveau des deux opérations est grande. Cependant, l'approche Average permet de déceler ce détail et favorise le premier service avec un degré d'appariement (4.83) supérieur à celui du deuxième service (4.33).

Je présente maintenant comment calculer avec beaucoup plus de précision le degré d'appariement final entre deux descriptions YASA. Pour ce faire, je prends un autre exemple de cas étudié et résumé dans la Table 5.3. Cette table présente les degrés d'appariements élémentaires entre un service requis et six services offerts. Ils ont tous une interface *Interface-Inf*, une opération *Operation-op1*, un input *Input-it1* et un output *Output-ot1*.

TABLE 5.3 – Exemples d'agrégation Min-Average

Services	Interface-Inf	Operation-op1	Input-it1	Output-ot1	Min	Average	Min-Average
1er Service	5	5	5	5	5	5	(5, 5)
2ème Service	5	5	5	4	4	4.83	(4, 4.83)
3ème Service	5	5	4	5	4	4.83	(4, 4.83)
4ème Service	5	5	4	4	4	4.67	(4, 4.67)
5ème Service	5	5	3	5	3	4.67	(3, 4.67)
6ème Service	5	5	5	3	3	4.67	(3, 4.67)

Considérons le 4ème et le 6ème service. Bien qu'ils aient une même valeur d'appariement Average (4.67), les deux services représentent deux situations différentes d'appariement. Les utilisateurs peuvent toujours exécuter le 4ème service sans avoir de problèmes. En fait, les concepts des inputs offerts sont plus généraux que ceux requis et les concepts des outputs offerts sont plus spécifiques que ceux requis. Cependant, l'exécution du 6ème service peut générer des outputs imprévus avec des concepts plus généraux que ce qui est attendu. C'est pour cela qu'il ne faut pas considérer que l'appariement de la requête, avec le 4ème et le 6ème service, est le même. J'ai démontré avec la première étude de cas que l'approche "Valeur minimale" d'agrégation n'est pas suffisante. De même pour l'approche Average dans cette deuxième étude de cas.

Pour remédier à toutes ces lacunes, je propose une nouvelle approche pour calculer le degré d'appariement final des descriptions YASA. Je définis un degré d'appariement final YASA par un couple de valeurs. La première représente le niveau d'appariement et la deuxième représente la valeur de l'appariement :

- Degré final d'appariement = (Niveau d'appariement , Valeur d'appariement)
- Le niveau d'appariement est calculé par l'approche d'agrégation "Valeur minimale",
- La valeur d'appariement est calculée par l'approche d'agrégation *Average*.

Les données agrégées sont, dans les deux cas, les degrés d'appariement élémentaire des éléments des services (interface, opération, input et output). Les degrés finaux d'appariement sémantique des six services sont calculés et affichés dans la dernière colonne "Min-Average" de la Table 5.3.

En prenant en compte, à la fois, le degré minimal et le degré moyen d'appariement, la précision du degré d'appariement global final augmente. Les couples de valeurs sont ordonnés d'abord par leurs valeurs d'appariement ensuite par leurs niveaux d'appariement. Dans la Table 5.3 et grâce à cette nouvelle approche, le 4ème service ayant le degré d'appariement (4, 4.67) est mieux évalué que le 6ème service ayant le degré d'appariement (3, 4.67), ainsi le problème identifié précédemment dans la Table 5.2 peut être résolu de la même manière.

5.4 Agrégations sémantiques : Cupid et Combinatory

L'Algorithme Min-Average suit une approche arithmétique pour l'agrégation. Cependant, les descriptions YASA peuvent être représentées sous forme d'arbres, ce qui me permet de m'inspirer de certaines approches d'appariement et d'agrégation sur les graphes, en particulier les arbres.

Une façon de calculer le degré d'appariement entre deux descriptions de services est de calculer le degré d'appariement entre deux arbres correspondant aux deux descriptions. En fait,

l'appariement entre arbres dépend essentiellement des degrés d'appariement entre les éléments correspondant de part et d'autre de chaque arbre et de la manière d'agréger ses valeurs afin d'obtenir le degré final global.

En appliquant mon mécanisme d'appariement sémantique élémentaire sur des nœuds d'arbres et en agrégeant les degrés d'appariement selon des algorithmes dédiés aux arbres, les résultats pourraient être intéressants et au profit de la découverte. Pour cette raison, dans cette section, je considère les approches Cupid [65] et Combinatory [33], deux approches reconnues en matière d'appariement et d'agrégation d'arbres. Et je propose deux algorithmes basés sur l'appariement et l'agrégation d'arbres, à savoir les algorithmes Cupid et Combinatory pour les services Web YASA.

Comme la plupart des algorithmes qui traitent des arbres (schémas, ontologies, etc.), deux variantes d'algorithmes d'appariement sémantiques Cupid et Combinatory combinent essentiellement deux processus :

- L'évaluation de la similarité entre les nœuds et les relations des arbres à comparer,
- L'agrégation de toutes les similarités pour le calcul de la similarité globale des arbres.

Les adaptations qui ont été réalisées sur les approches Cupid et Combinatory pour obtenir les variantes Cupid et Combinatory d'agrégation pour les descriptions YASA portaient sur :

- La transformation d'une description *YASA* en un arbre,
- La réalisation d'arbres à quatre niveaux : *service, interface, operations et input-output*,
- La délégation de l'appariement dans les opérations et les interfaces à l'appariement élémentaire (voir la section 5.2).

5.4.1 Agrégation sémantique Cupid

Dans cette section, je présente les principes de l'approche Cupid [65] pour l'appariement d'arbres, ensuite, je présente l'algorithme d'appariement et d'agrégation que nous avons développé en nous basant sur l'approche Cupid. Cet algorithme peut être utilisé aussi comme Average-Min pour agréger les valeurs d'appariements des descriptions de services Web YASA.

Principes de l'approche Cupid

L'algorithme Cupid présenté dans [65] est utilisé pour comparer des schémas assimilés à des arbres. Le sens de calcul part des feuilles vers la racine des deux arbres en cherchant une similarité nommée "similarité pondérée" (*weighted similarity*) pour chaque paire de nœuds de chacun des arbres. Le principe de cette similarité est le suivant : "Deux nœuds feuilles sont semblables si les nœuds au voisinage sont semblables. Et deux nœuds non-feuilles sont semblables si les nœuds feuilles de chacun des sous-arbres (pour lesquels elles sont racines) sont semblables".

La Figure 5.3 présente l'algorithme Cupid d'appariement de deux arbres S et T (ligne 1), avec :

- La similarité structurelle *Ssim* est une mesure de la similarité des voisinages des éléments.
- Les cœfficients de similarité linguistique (*Lsim*) sont calculés entre les éléments de schéma en comparant les termes extraits de leurs noms. On utilise, à cet effet, un thésaurus qui a des relations de synonymie et hyperonymie.
- La similarité pondérée (weighted) *Wsim* est une moyenne entre Lsim et Ssim.

```
1  CupidTreeMatch (SourceTree S, TargetTree T)
2  for each s in S, t in T where s,t are leaves
3      set Ssim (s,t) = datatype-compatibility(s,t)
4      S' = post-order(S), T' = post-order(T)
5      for each s in S'
6      for each t in T'
7          compute Ssim(s,t) = structural-similarity(s,t)
8          Wsim(s,t) = Wstruct * Ssim(s,t) + (1-Wstruct) * Lsim (s,t)
9          if Wsim(s,t) > thresholdHigh
10             increase-struct-similarity(leaves(s),leaves(t),C-inc)
11         if Wsim(s,t) < thresholdLow
12             decrease-struct-similarity(leaves(s),leaves(t),C-dec)
```

FIGURE 5.3 – Algorithme d'appariement de l'approche Cupid [65] .

– *Wsim = Wstruct * Wsim + (1-Wstruct) * Lsim*, avec Wstruct constante entre 0 et 1.

Le parcours se fait nœud par nœud en montant des feuilles à leurs nœuds parents (voir les lignes de 2 à 6). Une valeur initiale de similarité est affectée selon la compatibilité des données des nœuds comparés (ligne 3). Le parcours ascendant se fait des deux cotés à la fois (ligne 4). La similarité pondérée est calculée à la base de similarité structurelle *Ssim* et similarité linguistique (*Lsim*) (ligne 8). Si les deux éléments comparés sont très similaires, c'est-à-dire, si leur similarité pondérée dépasse le seuil *thresholdHigh* (ligne 9), on augmente la similarité structurelle (*ssim*) de chaque paire de feuilles dans les deux sous-arbres par le cœfficient *C-incr*, pour incrémenter et *ssim* ne doit pas dépasser 1 (ligne 10). En fait, les feuilles avec des ancêtres similaires ont des contextes similaires. Ainsi, cela renforce leur similarité structurelle. De même, si la similitude pondérée est inférieure au seuil *thresholdLow* (ligne 11), on diminue les similitudes structurelles de feuilles dans les sous-arbres par le cœfficient *C-dec*, pour décrémenter (ligne 12). La similarité linguistique demeure inchangée.

Algorithme d'agrégation Cupid pour les services YASA

L'Algorithme Cupid a été adapté pour l'agrégation de l'appariement tout en considérant les principes suivants d'appariement pour *YASA* :

– Les arbres sont constitués de 3 niveaux à apparier : interface, opération et input/output,
– L'appariement de deux interfaces, de deux opérations, de deux inputs ou de deux outputs est assuré par un appariement élémentaire sémantique (voir la section 5.2).

La Figure 5.4 présente l'algorithme :
– L'algorithme prend en entrée les deux arbres représentant les deux descriptions sémantiques de la requête et du service Web offert (ligne 1),
– Pour toutes les feuilles des deux arbres (les inputs/outputs), l'algorithme effectue des appariements sémantiques (lignes 2 - 4),
– Pour tous les nœuds parents des inputs/outputs (les opérations), l'algorithme effectue des appariements sémantiques (lignes 5 - 9), ensuite calcule les similarités structurelles (lignes 10 - 11),
– Selon l'approche Cupid, les similarités structurelles des sous-nœuds d'opérations peuvent être incrémentées ou décrémentées si elles dépassent les seuils : minimal *thresholdLow* et maximal *thresholdHigh* (lignes 12 - 15),

```
1  CupidAgregation (RequestedService R, OfferedService O){
2  for each r in R, o in O where r,o are leaves  {
3      set Ssim (r, o) = semanticSimilarity(r, o)
4      LeavesSim = leavesSimilarity(r, o) \\ inputs and outputs
5      R' = post-order(R), O' = post-order(O)
6      for each r in R'
7      for each o in O'
8      set Ssim (r, o) = semanticSimilarity(r, o)
9      OperationsSim = operationSimilarity(r, o, LeavesSim){
10         compute Ssim(r, o) = structural-similarity(r, o)
11         Wsim(r, o) = Wstruct * Ssim(r,o) + (1-Wstruct) * semanticSimilarity (r, o)
12         if Wsim(r, o) > thresholdHigh
13             increase-struct-similarity(leaves(r), leaves(o), C-inc)
14         if Wsim(r, o) < thresholdLow
15             decrease-struct-similarity(leaves(r), leaves(o), C-dec)
16      }
17        R'' = post-order(R'), O'' = post-order(O')
18        for each r in R''
19        for each o in O''
20        set Ssim (r, o) = semanticSimilarity(r, o)
21        ServiceAggregatedSim = interfaceSimilarity(r, o, OperationsSim){
22            compute Ssim(r, o) = structural-similarity(r, o)
23            Wsim(r,o) = Wstruct * Ssim(r, o) + (1-Wstruct) * semanticSimilarity (r, o)
24         if Wsim(r, o) > thresholdHigh
25             increase-struct-similarity(leaves(r), leaves(o), C-inc)
26         if Wsim(r, o) < thresholdLow
27             decrease-struct-similarity(leaves(r), leaves(o), C-dec)
28      }
29    return ServiceAggregatedSim
30 }
```

FIGURE 5.4 – Algorithme d'agrégation Cupid pour les services YASA.

- Pour tous les nœuds parents des opérations (les interfaces), l'algorithme effectue des appariements sémantiques (lignes 17 - 21), ensuite calcule les similarités structurelles (lignes 17 - 23),
- Selon l'approche Cupid, les similarités structurelles des sous-nœuds d'interfaces peuvent être incrémentées ou décrémentées suivant les seuils (lignes 24 - 27),
- Le degrés final d'appariement agrégé tout au long de l'algorithme est retourné (lignes 4 , 9 , 21 et 29).

5.4.2 Agrégation sémantique Combinatory

Principes de l'approche Combinatory

Dans l'approche dite systématique [33] qui utilise l'agrégation combinatoire, les arbres sont comparés d'une manière récursive et combinatoire. L'algorithme considère que les similarités syntaxiques des nœuds ainsi que des arcs ont été déjà calculées. En partant des racines des arbres, on applique la formule suivante :

$$SoG(n_Q, n_R) = w(n_Q, n) \cdot sim_n(n_Q, n_R) + \max_{\substack{for\ all \\ combinations\ j}} \{\sum w(n_Q, j) \cdot sim_a(a_Q^j, a_R^j) \cdot SoG(n_Q^{a_Q^j}, n_R^{a_R^j})\}$$

SoG dénote la similarité des graphes (*Similarity Of Graphs*) entre un graphe RDF ayant comme nœud d'entrée n_Q (*Query*) et un graphe RDF ayant comme d'entrée n_R (*Resource*). sim_n dénote une similarité entre deux nœuds et sim_a dénote une similarité entre deux arcs a_Q et a_R dérivés respectivement des nœuds n_Q et n_R. $w(n_Q,n)$ et $w(n_Q,j)$ sont les valeurs affectées au

nœud et arc en cours. La fonction *max* et *SoG* assure le parcours des sous-arbres et la sélection parmi toutes les combinaisons possibles, de celle qui a le maximum de similarité cumulée.

Algorithme d'agrégation Combinatory pour les services YASA

Dans le cadre de l'agrégation de l'appariement sémantique, l'approche combinatoire est intégrée et implémentée sans modifier le principe de l'algorithme. A l'implémentation, la seule petite différence avec ce qui a été fait dans [33] est que les valeurs de l'appariement élémentaire n'étaient pas pré-calculées à l'avance.

L'adaptation de l'approche d'agrégation Combinatory pour l'appariement de *YASA* a été faite comme indiqué dans la suite.

La Figure 5.5 illustre des arbres représentants des services Web YASA. "q" dénote le service offert et "r" dénote le service requis, avec $a_q i$ et $a_r j$ sont les arcs qui relient les racines (les interfaces) $n_q 1$ et $n_r 1$ à leurs fils directs (les opérations). Et $n_q i$ et $n_r j$ sont les fils directs de $n_q 1$ et $n_r 1$.

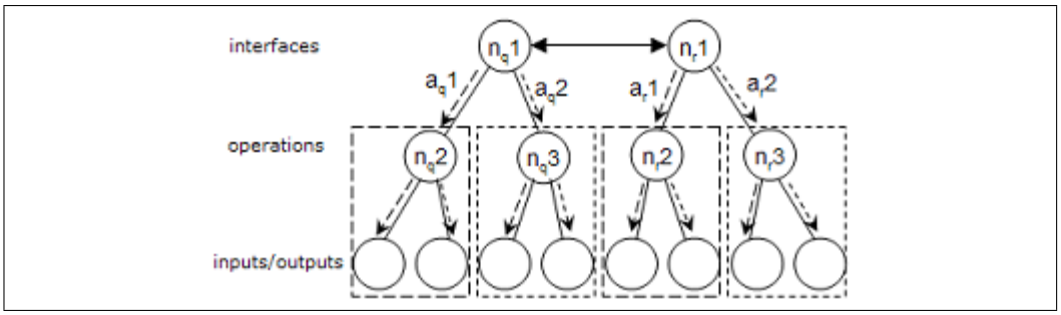


FIGURE 5.5 – Agrégation combinatoire et récursive sur des arbres de services Web YASA.

Dans un premier temps, l'algorithme (voir Figure 5.6) assure le calcul des degrés d'appariement entre les interfaces (lignes 3-6), ensuite, à travers les arcs, il bascule sur le calcul des degrés d'opérations (lignes 7-10). Puis, l'algorithme calcule les degrés d'appariement sémantique des inputs et des outputs (lignes 12-21). Dans un deuxième temps, l'algorithme assure l'agrégation des degrés d'appariement maximum entre les inputs et des outputs (lignes 23-26), ensuite, à travers les arcs, il bascule sur l'agrégation des degrés maxima d'opérations (lignes 29-31). Enfin, l'algorithme agrège les degrés d'appariement sémantique maximum des interfaces (lignes 33-34).Le degré d'appariement agrégé est retourné par l'algorithme (ligne 36).

5.5 Conclusion

Dans ce chapitre, j'ai présenté le principe de l'appariement global ainsi que mon approche d'appariement sémantique élémentaire. Ensuite, pour l'agrégation des valeurs d'appariement élémentaires et le calcul du degré d'appariement global de deux descriptions YASA, j'ai présenté les trois algorithmes d'agrégation. Le premier algorithme Min-Average [11] est basé sur une approche arithmétique. Et comme les descriptions YASA peuvent être représentées sous forme d'arbres, je me suis inspiré de certaines approches d'appariement et d'agrégation sur les graphes, en particulier les arbres afin de proposer deux autres algorithmes Cupid et Combinatory pour agréger les degrés sémantiques élémentaires. La complexité des deux algorithmes adaptés est reste la

```
CombinatoryAgregation (RequestedService R, OfferedService Q){
matchInterfaces(R, Q){
for each  ri in interfaces(R)
for each  qi in interfaces(Q){
    compute  matchInterfaces(RI, QI)   {
       for each  ro in operations(RI)
       for each  qo in operations(QI){
                compute  matchOperations(RI, QI)
                for each  rInp in inputs(RO)
                for each  qInp in inputs(QO){
                      compute matchInputs(RO,QO){
                           degreeInput = semanticComparator(rInp,qInp)
                           checkMaxInput (degreeInput)}
               for each  rOut in outputs(RO)
               for each  qOut in outputs(QO)
                  compute  matchOutputs(RO,QO){
                      degreeOutput = semanticComparator(rOut,qOut)
                      checkMaxOutput (degreeOutput)}
     degreeOperation = (similarityCoefficient * (semanticComparator(ri, qi))
    + (1-similarityCoefficient) * Max(getMax(matchInputs(ro,qo), getMax(matchOutputs(ro, qo)))
     checkMaxOperation (degreeOperation)}
 degreeInterface = (similarityCoefficient * (semanticComparator(ri, qi))
   + (1-similarityCoefficient) * getMax(matchOperations(ri, qi)))
 checkMaxInterface(degreeInterface)   }
ServiceAggregatedSim = similarityCoefficient * (semanticComparator(ri, qi))
   + (1-similarityCoefficient) * getMax(matchInterfaces (ri, qi))
return ServiceAggregatedSim
}
```

FIGURE 5.6 – Algorithme d'agrégation Combinatory pour les services YASA.

même que les algorithmes d'origines, les adaptations faites n'influencent pas leurs complexités. Le chapitre suivant 6 présente comment intégrer de tels mécanismes d'appariement et d'agrégation au sein d'une découverte de service dans un annuaire sémantique de services Web.

Chapitre 6

L'annuaire des services Web sémantiques

Sommaire

6.1 Introduction

Afin de vérifier et valider mes propositions en termes de description, d'appariement et de découverte de services Web sémantiques, un système adéquat devait être conçu et déployé en mettant en œuvre mes choix. La construire d'un annuaire de services Web permet de vérifier et valider mes propositions. L'annuaire permettra de réfléchir et de mettre en œuvre une approche de stockage de la description sémantique *YASAWSDL* tout en gardant à l'esprit son statut d'extension du standard *WSDL* et de SAWSDL. Se baser sur des modèles d'annuaires dédiés au standard *WSDL* est toujours avantageux. L'annuaire permettra, aussi, de mettre en œuvre mon approche d'appariement sémantique et de découverte automatique.

Dans ce chapitre, je décris l'architecture de l'annuaire, ces principaux composants et comment les services Web décrits en *YASAWSDL* sont stockés et découverts. L'annuaire des services Web sémantiques consiste en un ensemble de composants de publication, d'appariement et de découverte. Ces composants s'intègrent dans une architecture de services Web à séman-

tique fonctionnelle. L'architecture a été conçue, implémentée et validée dans le cadre du projet SemEUsE [1].

Ce chapitre est organisé comme suit : je commence par présenter l'architecture de l'annuaire des services Web sémantiques (Section 6.2). Ensuite, je décris comment l'annuaire assure la phase de publication et de stockage des descriptions sémantiques de services (Section 6.3). Finalement, je me focalise sur la phase d'appariement et de découverte (Section 6.4).

6.2 Architecture de l'annuaire des services Web sémantiques

6.2.1 Composants de l'architecture

L'architecture de l'annuaire est présentée dans la Figure 6.1. Cet annuaire est conçu et implémenté afin d'intégrer un bus de service sémantique d'un projet national de recherche (SemEUsE [91]). Le projet a pour objectif de fournir une architecture de services Web sémantiques, sensible au contexte. Il est basé sur deux aspects : un premier aspect de conception guidé par des modèles théoriques de services sémantiques et un deuxième aspect de *runtime* guidé par un bus dynamique de services sémantiques (voir [91] pour plus de détails).

L'annuaire permet, dans un premier temps, la publication, le stockage des descriptions sémantiques de services. Dans un deuxième temps, il permet, grâce à leurs descriptions sémantiques, la découverte de ces services. Pour ce faire, l'annuaire est doté des composants suivants :

- le registre sémantique fonctionnel : *SemanticFunctionalRepository*,
- le matchmaker sémantique fonctionnel : *SemanticFunctionalMatching*,
- le registre syntaxique : *SyntacticRepository*,
- l'annuaire sémantique : *SemanticRegistry*,
- le composant d'administration : *Admin*,
- le courtier : *Trader* (ou *Mediator*).

6.2.2 Rôles des composants

Dans cette section, je présente les composants *SemanticRegistry, SyntacticRepository, SemanticFunctionalRepository, SemanticFunctionalMatching, Trader* et *Admin*. Cette section présente aussi leurs principales fonctionnalités, leurs interfaces et les interactions avec d'autres composants.

SemanticRegistry

Grâce à l'annuaire sémantique (*SemanticRegistry*), les fournisseurs de services publient les descriptions d'interfaces de leurs services offerts aux consommateurs. Ce composant fournit les mécanismes nécessaires pour déclencher le stockage des descriptions des services à la fois dans le registre syntaxique (*SyntacticRepository*) et le registre sémantique fonctionnel (*SemanticFunctionalRepository*). A l'instar de l'approche UDDI, le registre syntaxique sert à stocker les

1. SemantiquE pour bUs de sErvices (http ://www.semeuse.org) : Projet financé par l'Agence Nationale de Recherche intégrant la sémantique, le contexte et la sécurité pour la publication, la découverte, l'invocation et la composition de services Web dans un bus

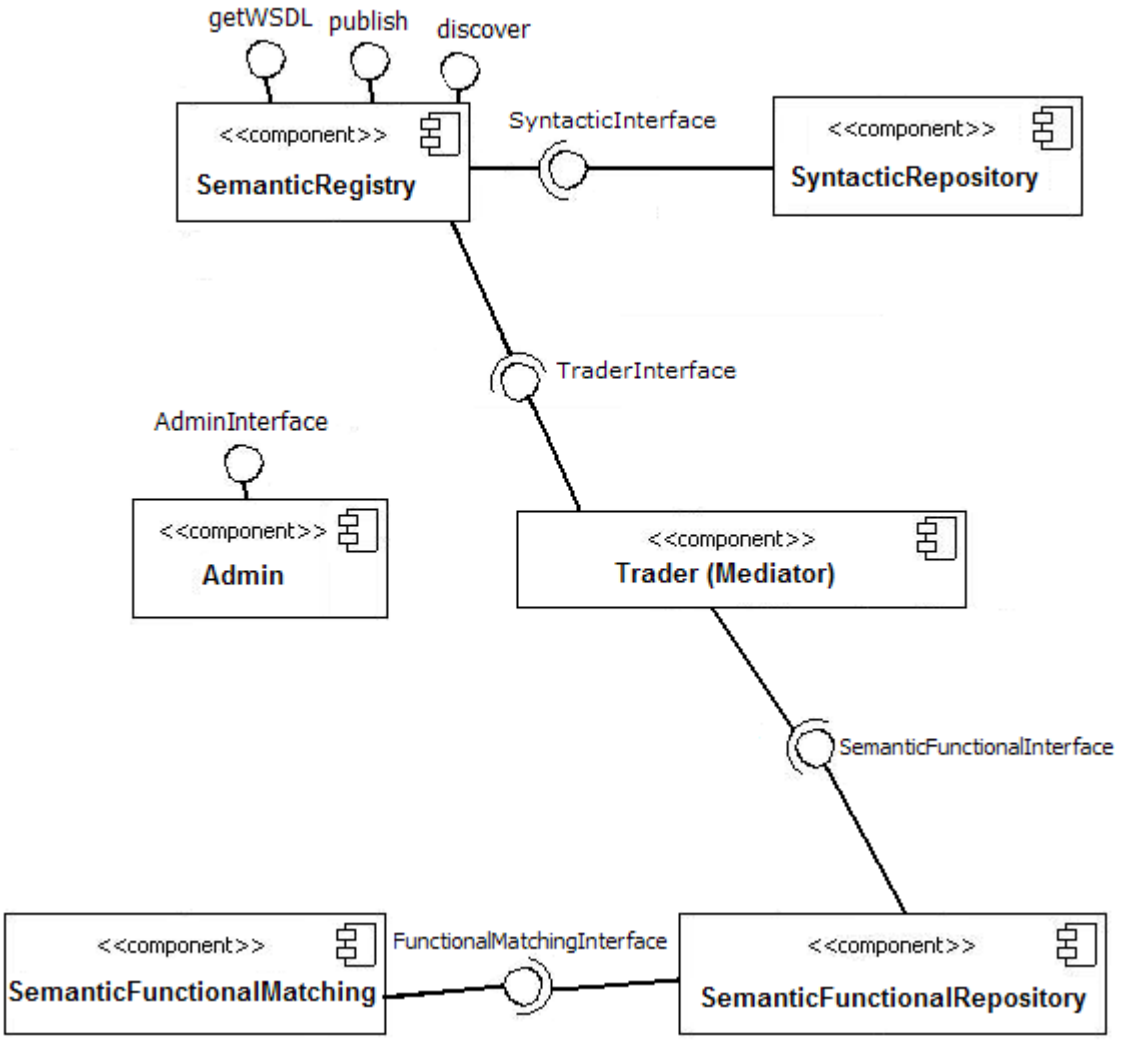


FIGURE 6.1 – Architecture de l'annuaire des services Web sémantiques.

descriptions des services offerts par les fournisseurs uniquement d'un point de vue syntaxique. Les données sémantiques ne sont visibles et utiles que grâce au registre sémantique fonctionnel qui est conçu dans cet objectif. Pour cela, le registre sémantique fonctionnel fournit des capacités de stockage des descriptions sémantiques des services déjà publiés dans le registre syntaxique.

L'annuaire sémantique offre trois interfaces : "*publish*", "*getWSDL*" et "*discover*". Elles assurent, respectivement, le déclenchement de la publication, la récupération et la découverte des descriptions de services. Ce composant interagit directement avec le registre syntaxique (*SyntacticRepository*) et le courtier (*Trader*). Des détails sur ces interfaces et ces interactions sont donnés dans les paragraphes suivants.

SyntacticRepository

Au moment de la publication, le registre syntaxique (*SyntacticRepository*) stocke la description complète de service et retourne un identifiant unique pour chaque service publié. Ce même

identifiant sert de référence pour publier la description sémantique du service dans le registre sémantique fonctionnel. Ceci permettra de garder une trace et de maintenir une cohérence tout au long de l'architecture de l'annuaire. Tout le processus veillant sur cette gestion se déclenche à partir de l'interface "*publish*" du composant *SemanticRegistry* au moment de la publication d'une description *YASAWSDL*.

De même, l'annuaire sémantique fournit aux consommateurs de services les mécanismes nécessaires pour déclencher, auprès du registre syntaxique, la récupération d'un contenu de service déjà publié en utilisant les identifiants uniques établis au moment de la publication. Tout le processus qui gère ceci se déclenche à partir de l'interface "*getWSDL*" du composant *SemanticRegistry* en lui précisant simplement l'identifiant unique d'une description *YASAWSDL* à récupérer.

Le registre syntaxique interagit uniquement avec l'annuaire sémantique, grâce à son "interface syntaxique" (*SyntacticInterface*).

SemanticFunctionalRepository

Pour le stockage et la découverte des descriptions sémantiques, il faut passer par le registre sémantique fonctionnel (*SemanticFunctionalRepository*). En effet, l'annuaire sémantique fournit aussi aux consommateurs de services les mécanismes nécessaires pour déclencher via le *Trader* et auprès du registre sémantique fonctionnel, la découverte sémantique automatique de services en utilisant une description sémantique requise.

En fait, le registre sémantique fonctionnel (*SemanticFunctionalRepository*) fournit, grâce à son interface *SemanticFunctionalInterface*, des capacités de récupération des descriptions sémantiques de services qui y sont déjà publiés. Tout le processus veillant sur cette gestion se déclenche à partir de l'interface "*discover*" du composant *SemanticRegistry* au moment de la découverte d'une description *YASAWSDL*.

Au cours du stockage de descriptions, le registre sémantique fonctionnel interagit avec le *Trader* grâce à l'interface *SemanticFunctionalInterface*. Et au cours de la découverte, le registre sémantique fonctionnel interagit avec le matchmaker sémantique fonctionnel (*SemanticFunctionalMatching*) afin que ce dernier réalise les appariements sémantiques.

SemanticFunctionalMatching

Au niveau du matchmaker sémantique fonctionnel (*SemanticFunctionalMatching*), la découverte effectue des appariements sémantiques basés sur les ontologies. L'appariement se fait sur l'aspect fonctionnel en sélectionnant les services candidats qui correspondent à la requête de l'utilisateur. Comme précédemment expliqué dans le chapitre 5, ce composant assure l'appariement entre les éléments décrits par les concepts requis (tâches requises par un utilisateur) et les éléments décrits par les concepts offerts (capacités de services publiées par un fournisseur).

L'interface *SemanticMatchingInterface* permet au registre sémantique fonctionnel (*SemanticFunctionalRepository*) d'interagir avec le matchmaker sémantique fonctionnel (*SemanticFunctionalMatching*).

Trader

Le courtier (*Trader*), au cœur de l'architecture, assure toute interaction entre l'annuaire sémantique (*SemanticRegistry*) et le registre sémantique fonctionnel (*SemanticFunctionalRepository*). Il joue le rôle d'un médiateur (*Mediator*) entre les différents composants de l'architecture.

L'interface de courtage (*TraderInterface*) permet à l'annuaire sémantique (*SemanticRegistry*) d'interagir avec le courtier (*Trader*). En fait, le courtier a été ajouté à l'architecture pour faire la médiation avec des matchmakers non fonctionnels (*QoS matchmaker, Security matchmaker,...*) qui existent dans le projet SemEUsE, en plus du matchmaker sémantique fonctionnel présenté dans mes travaux.

Admin

Le composant d'administration (*Admin*) gère les droits d'accès des fournisseurs et des consommateurs de services aux registres par le biais d'une interface de contrôle d'accès. Grâce à l'interface *AdminInterface*, on peut vérifier également la disponibilité des services publiés dans les registres et offrir des statistiques concernant ces services aux fournisseurs et consommateurs intéressés.

Récapitulatif

Pour résumer, les interfaces les plus importantes sont :

- l'interface syntaxique (*SyntacticInterface*) : elle permet de faire appel au processus de stockage et de récupération des descriptions *WSDL* des services auprès du registre syntaxique,
- l'interface sémantique fonctionnelle (*SemanticFunctionnalInterface*) : elle permet de faire appel au processus de stockage, de recherche et de récupération des descriptions sémantiques de services auprès du registre sémantique fonctionnel,
- l'interface d'appariement sémantique (*SemanticMatchingInterface*) : elle permet de faire appel au processus d'appariement des descriptions sémantiques YASA auprès du matchmaker sémantique.

Les descriptions de services Web YASA peuvent être publiées sans aucune modification dans *SyntacticRepository*. Cependant, il est nécessaire d'enregistrer les informations sémantiques provenant des descriptions de services Web YASA dans des éléments à part dans le *SemanticFunctionalRepository*. Ceci permet de différencier la découverte syntaxique de la découverte sémantique de services. Ces derniers aspects sont assurés par les composants *SyntacticRepository* et *SemanticFunctionalMatching*.

Le reste de ce chapitre présente comment l'annuaire des services Web sémantiques assure les phases de publication et de découverte conçues autour des propriétés fonctionnelles.

6.3 Phase de Publication

La phase de publication implique des interactions entre certains composants. Ces interactions ainsi que les détails concernant la publication des services et le stockage de leur sémantique sont présentés dans les paragraphes de cette section.

6.3.1 Interactions pour la publication

L'interface de publication permet à un fournisseur de publier les descriptions de ses services. La description des services sémantiques dans l'annuaire est spécifiée en *YASAWSDL*. L'annuaire accepte également des descriptions *WSDL* ou SAWSDL. La Figure 6.2 présente un diagramme de séquences pour la phase de publication.

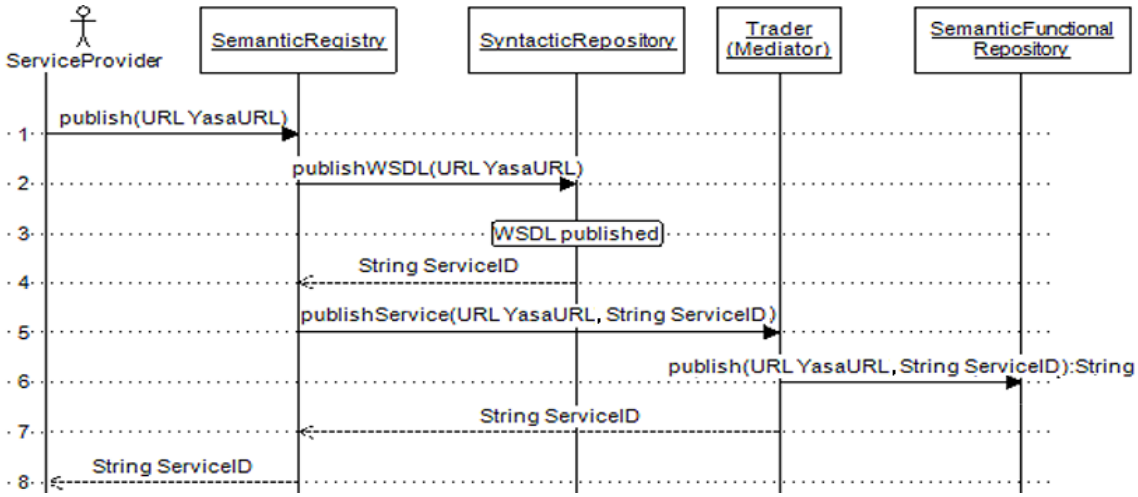


FIGURE 6.2 – Diagramme de séquences pour la phase de publication.

La phase de publication donne lieu à des interactions entre le registre sémantique fonctionnel (*SemanticFunctionalRepository*), le registre syntaxique (*SyntacticRepository*), l'annuaire sémantique (*SemanticRegistry*) et le courtier (*Trader*).

La phase de publication débute quand un fournisseur de service (*ServiceProvider*) envoie à l'annuaire sémantique une URL identifiant la description de son service. Pour cela, il utilise la méthode "*publish*" (Figure 6.2 , étape 1). L'annuaire sémantique appelle la méthode "*publishWSDL*" auprès du registre syntaxique (Figure 6.2 , étape 2). Pour chaque *WSDL* publié, le registre syntaxique (*SyntacticRepository*) lui associe une chaîne de caractères unique, appelée identifiant de service : *ServiceID* (Figure 6.2 , étape 3).

Le *ServiceID* est retourné au *SemanticRegistry* (Figure 6.2 , étape 4). Le *SemanticRegistry* renvoie la description de service et son identifiant unique au *Mediator* qui les renvoie à son tour au registre sémantique fonctionnel.

Ce dernier composant analyse la description ; les éléments sémantiques fonctionnels de la description sont extraits et préparés afin d'être stockés. Le même identifiant retourné par le registre syntaxique servira de même comme identifiant et référence de la description sémantique dans le registre sémantique fonctionnel (Figure 6.2 , étape 5 et 6). Le *Trader* confirme le stockage

de la description sémantique à l'annuaire sémantique qui à son tour le confirme au fournisseur en lui renvoyant le *ServiceID* (Figure 6.2 , étape 7 et 8).

6.3.2 Stockage de la sémantique

Le registre sémantique fonctionnel est conçu dans l'objectif de rendre visibles et utiles les données sémantiques transparentes au niveau du registre syntaxique. Ainsi, le registre sémantique fonctionnel fournit des capacités de stockage de l'aspect sémantique des descriptions de services Web.

En effet, le registre sémantique fonctionnel extrait les informations sémantiques de la description *YASAWSDL*, ensuite, il crée à partir de ces données une description *RDF*. La description *RDF* ne reprend pas exactement tous les éléments de la description originelle. J'explique la procédure de transformation et l'intérêt de mes choix à ce niveau en détails dans le paragraphe suivant. Pour le moment, je précise uniquement que les informations sémantiques récupérées sont transformées en triplets *RDF*. Chaque donnée sémantique (concept de service ou concept de domaine) est attachée à l'élément *WSDL* qu'elle annote dans la description *YASAWSDL* (e.g. interface, input, etc).

Dans la description *RDF*, le registre sémantique fonctionnel insère comme référence l'identifiant unique du service auprès du registre syntaxique. Ceci permettra de garder le lien entre la sémantique stockée et le service publié. Ainsi, au moment de la découverte, il sera très facile de retrouver la description syntaxique d'un service Web correspondant à la sémantique fonctionnelle requise. Entre autre, ce mécanisme permet aussi de garder une trace auprès de l'utilisateur et de maintenir la cohérence entre les registres de l'annuaire. Une fois les transformations faites et l'identifiant unique ajouté, les descriptions *RDF* sont prêtes à être stockées dans une base de connaissances (voir Figure 6.3).

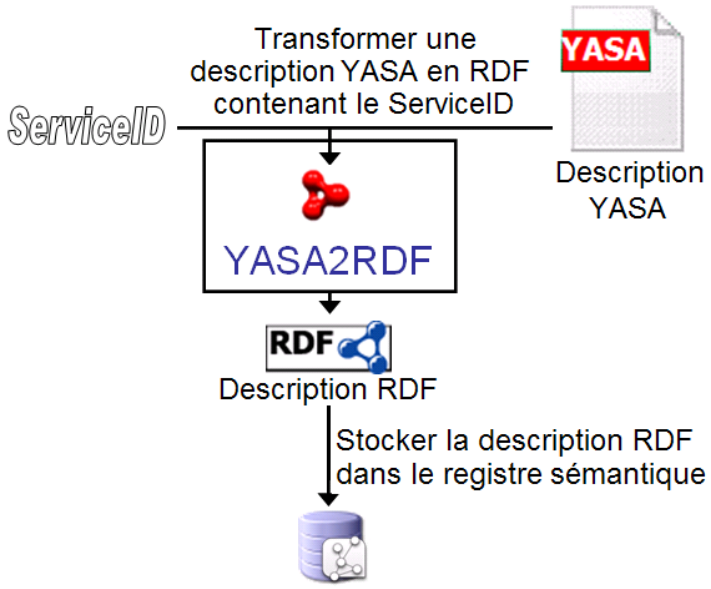


FIGURE 6.3 – Transformation de la description *YASAWSDL* en description sémantique *RDF*.

Dans la suite, j'explique étape par étape comment créer une description RDF à partir d'une description *YASAWSDL*. En première étape, considérons l'exemple de description *YASAWSDL*

```
<interface name = "FileManagement">
   <operation name= "openRentalFile"/>

       <input messageLabel= "custId"/>
       <input messageLabel= "carId"
         serviceconcept= "precondition" modelreference= "CarId"/>
       <output messageLabel= "fileNb"/>

   </operation>

   <operation name= "closeRentalFile"
     serviceconcept= "effect" modelreference= "RentalClosing"/>

       <input messageLabel= "fileNb"/>
       <output messageLabel= "success"/>

   </operation>
</interface>
```

FIGURE 6.4 – Exemple de description *YASAWSDL*.

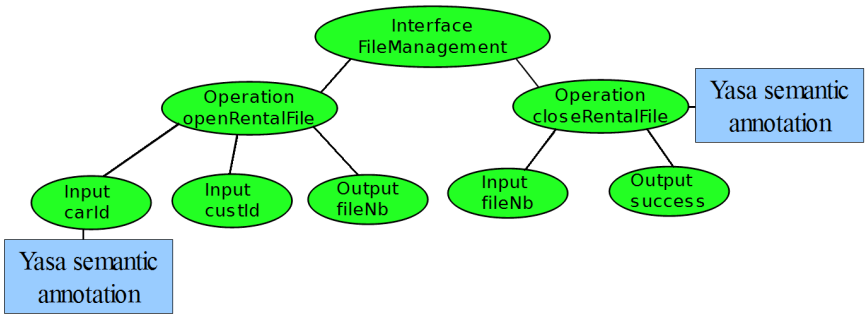


FIGURE 6.5 – Arbre sémantique complet de description *YASAWSDL*.

de la Figure 6.4) et son "arbre complet" représentatif de la Figure 6.5. D'après l'arbre représentatif complet qui est construit dans une première étape à partir de la description *YASAWSDL*, seuls deux éléments sont annotés. Les autres ne portent aucune donnée sémantique. Il est clair qu'il serait plus sensé d'éviter de les réintégrer dans la nouvelle description sémantique RDF, puisqu'ils sont stockés dans l'annuaire syntaxique. Pour cela, dans la deuxième étape, je propose de construire un nouveau arbre dit "arbre sémantique réduit" en ne gardant que les chemins à partir de la racine et allant vers des nœuds avec des annotations sémantiques (voir Figure 6.6).

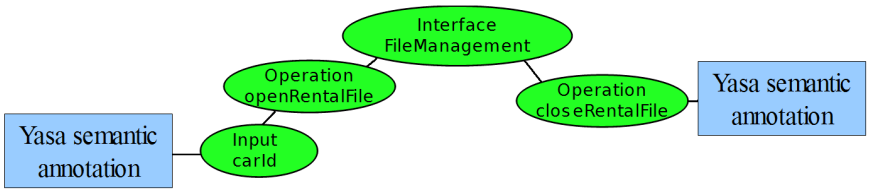


FIGURE 6.6 – Arbre sémantique réduit de description *YASAWSDL*.

Dans la troisième étape, je transforme l'arbre sémantique réduit en triplets RDF qui réutilisent les noms des éléments WSDL non supprimés, les concepts de l'ontologie de services Web et les concepts de domaines (correspondants à chaque annotation). En effet, chaque nœud non supprimé se transforme en ressource RDF. Les noms des ressources remplacent les termes entre "[ ]" de la Figure 6.7 qui représente le graphe RDF générique.

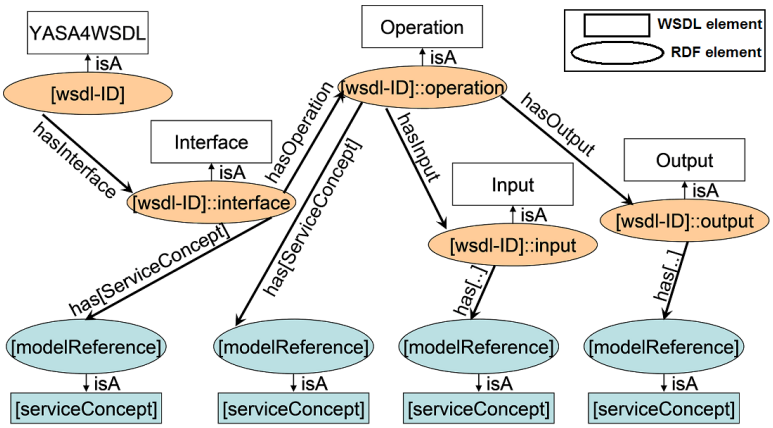


FIGURE 6.7 – Le graphe RDF générique.

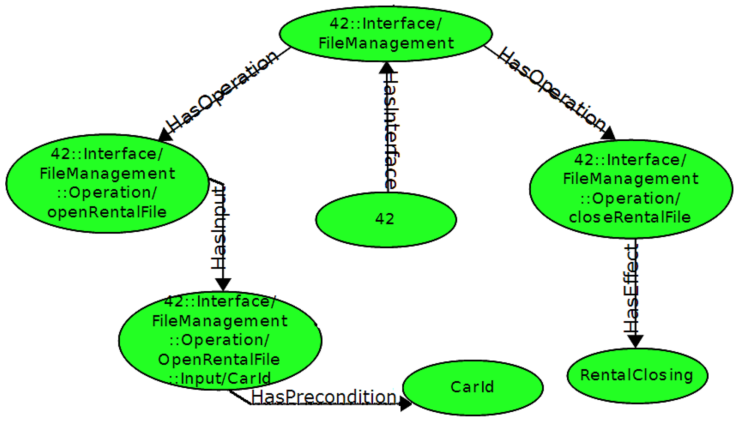


FIGURE 6.8 – Graphe RDF d'illustration.

```
<Description::4028819421323a340121323a794c0008/interface::reservationInterface2/interfaceOperation::opreserve2[
InterfaceOperation],hasEffect,http://stromboli.it-sudparis.eu/~chabeb_y/tourism.owl#CreditCard[Effect]>

<Description::4028819421323a340121323a794c0008/interface::reservationInterface2/interfaceOperation::opreserve2[
InterfaceOperation],hasInput,Description::4028819421323a340121323a794c0008/interface::reservationInterface2/
interfaceOperation::opreserve2/inputMessage::In[Input]>
```

FIGURE 6.9 – Extrait de code d'une description RDF.

La Figure 6.8 illustre le graphe RDF et la Figure 6.9 présente un extrait de la description RDF.

6.4 Phase de découverte

Cette phase déclenche entre les composants des interactions concernant l'appariement et la découverte des services Web. Les détails de ces interactions sont présentés dans les paragraphes de cette section.

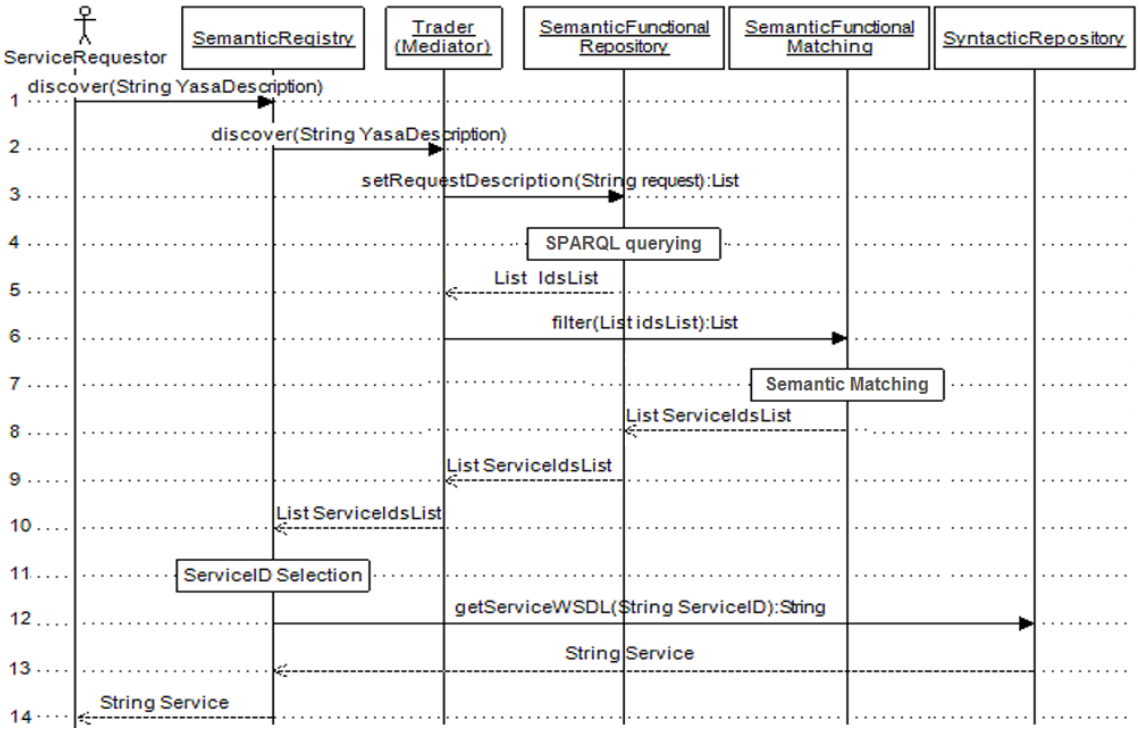


FIGURE 6.10 – Diagramme de séquences pour la phase de découverte.

6.4.1 Interactions pour la découverte

Les interactions pour la découverte concernent : l'annuaire sémantique (*SemanticRegistry*), le courtier (*Mediator*), le registre sémantique fonctionnel (*SemanticFunctionalRepository*), le matchmaker sémantique fonctionnel (*SemanticFunctionalMatching*) et le registre syntaxique (*SyntacticRepository*). La Figure 6.10 présente un diagramme de séquences pour la phase de découverte.

L'interface de découverte permet à un consommateur de trouver un ou plusieurs services qui correspondent à sa demande. Il suffit que le consommateur soumette une description YASAWSDL d'un service donné ou tout simplement une description incomplète spécifiant uniquement des éléments WSDL annotés sémantiquement en YASAWSDL.

Ainsi, une requête peut être une simple description fonctionnelle sémantique spécifiée en YASAWSDL sans les parties *binding* et *service*, juste une partie *interface/portType*. La requête est vue comme une description abstraite de service Web avec des annotations sémantiques sur les éléments WSDL : *interface, operation, input* et *output*.

La phase de découverte débute quand un consommateur de service (*ServiceRequestor*) envoie à l'annuaire sémantique (*SemanticRegistry*) sa requête de service Web. Pour cela, il utilise une méthode "*discover*" (Figure 6.10, étape 1). A son tour, l'annuaire sémantique renvoie la requête en appelant une méthode au même nom auprès du *Trader* (Figure 6.10, étape 2). Ce

dernier renvoie le YASAWSDL requis au registre sémantique fonctionnel pour l'apparier aux services offerts. Pour homogénéiser le format du YASAWSDL requis par rapport aux descriptions sémantique RDF stockées, le registre sémantique fonctionnel transforme le YASAWSDL requis en une requête RDF sémantique (Figure 6.10, étape 3). Pour cela, j'utilise une requête SPARQL pour interroger la base de données et sélectionner uniquement les services qui répondent à la requête. L'appariement de la sémantique de la requête avec un groupe de sémantiques de services se base sur la ressemblance de graphes RDF. Cet appariement assure l'extraction de services proche structurellement en termes d'éléments WSDL et d'annotations sémantiques puisque chacune de ces données est représentée dans le graphe RDF (Figure 6.10, étape 4).

Une fois sélectionnées, les descriptions sémantiques adéquates contiennent les identifiants de leurs services Web déjà publiés au registre syntaxique. La liste des identifiants est récupérée et retournée au *Trader* (idsList en Figure 6.10, étape 5). Ce dernier renvoie la liste au matchmaker sémantique fonctionnel (*SemanticFunctionalMatching*) afin qu'il puisse apparier sémantiquement le YASAWSDL requis aux YASAWSDL offerts. Pour cela, le *Trader* appelle la méthode "filter" (Figure 6.10, étape 6). Le matchmaker sémantique fonctionnel ne doit garder que les services Web sémantiquement pertinents. Il réalise l'appariement sémantique Average-Min entre la requête et chaque service Web appartenant à la liste (Figure 6.10, étape 7). Seuls les services qui se révèlent sémantiquement pertinents sont gardés et une nouvelle liste contenant leurs identifiants est retournée au *Trader* (ServiceIdsList en Figure 6.10, étape 8).

Une fois validée par le matchmaker sémantique fonctionnel, la liste des services Web est renvoyée à l'annuaire sémantique (*SemanticRegistry*) (Figure 6.10, étape 9). Il sélectionne automatiquement le service Web ayant la meilleure évaluation selon l'approche Average-Min (Figure 6.10, étape 10). Ensuite, il récupère grâce à l'identifiant du service sélectionné la description WSDL (YASAWSDL) depuis le registre syntaxique (*SyntacticRepository*) (Figure 6.10, étapes 11, 12 et 13). Finalement, l'annuaire sémantique (*SemanticRegistry*) renvoie la description du service Web au consommateur de service (*ServiceRequestor*) (Figure 6.10, étape 14).

6.4.2 Appariement RDF : une phase de pré-selection

Afin d'améliorer la découverte en termes de pertinence et de performance, je propose de procéder en deux phases. Une première phase de pré-sélection de description sémantique basée sur un appariement RDF, ensuite, une deuxième phase d'appariement sémantique fonctionnel.

L'appariement sémantique fonctionnel est déjà présenté dans le chapitre 5. Je donne dans la suite plus d'informations concernant l'appariement RDF.

Ces deux grandes phases proposées se complètent bien qu'elles soient rattachées à deux approches différentes en termes de découverte. La première phase est caractérisée par un appariement basé sur l'appariement des descriptions RDF (RDF-based Matching) entre la requête et les services Web offerts. La deuxième phase est caractérisée par un appariement basé sur les annotations sémantiques des descriptions YASAWSDL (YASA-based Matching) entre la requête et les services Web publiés.

La découverte démarre par la soumission d'une description YASAWSDL de la requête à l'annuaire sémantique. De son coté, le registre sémantique fonctionnel stocke des descriptions RDF. Pour réussir l'appariement entre requête et descriptions RDF, la description YASAWSDL requise est transformée en une requête RDF sémantique. Une vue globale des phases de la découverte est illustrée dans la Figure 6.11.

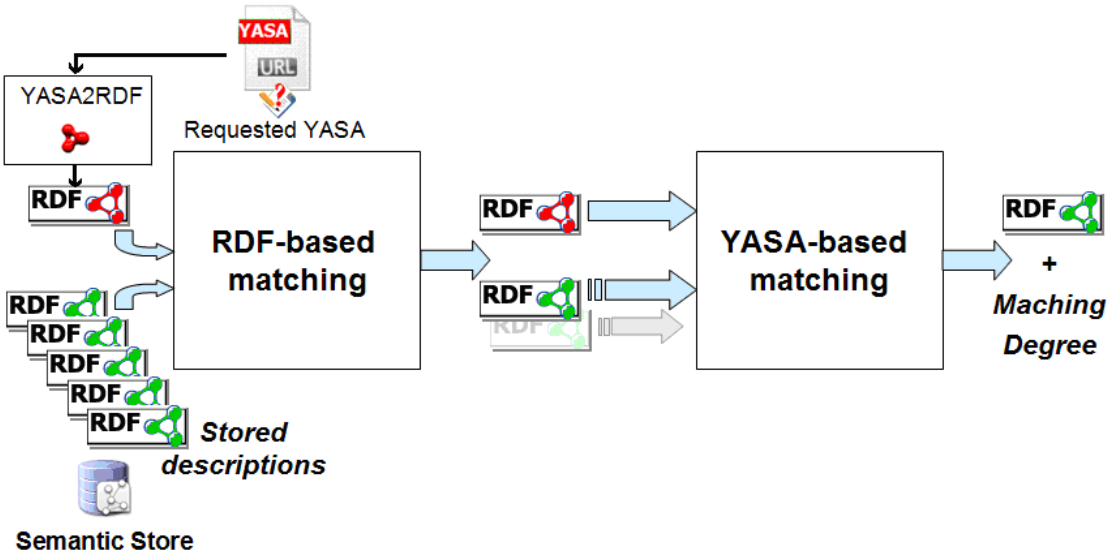


FIGURE 6.11 – Vue globale des phases de la découverte.

En se basant sur les descriptions RDF stockées, vues souvent comme des graphes, je propose de ne pré-sélectionner que celles qui correspondent au "graphe" de la requête. Cette pré-sélection vise à alléger la tâche d'appariement sémantique fonctionnel jugée souvent comme coûteuse. Pour cela, je construis une requête SPARQL à partir de la requête utilisateur, ensuite j'interroge la base de données des descriptions pour sélectionner uniquement les services qui m'intéressent d'un point de vue structurel.

En effet, cet appariement se base sur la ressemblance de graphes RDF. Il permet ainsi la récupération du groupe de services proches structurellement en termes d'éléments WSDL et d'annotations sémantiques puisque chacune de ces données est représentée dans le graphe RDF. Un tel choix est très important voire vital afin d'éviter de comparer la requête à tous les services Web disponibles. Une fois ce sous-ensemble de services Web sélectionnées, le match-maker sémantique fonctionnel effectue les appariements sémantiques fonctionnels nécessaires entre le YASAWSDL requis et les YASAWSDL offerts (voir Figure 6.12).

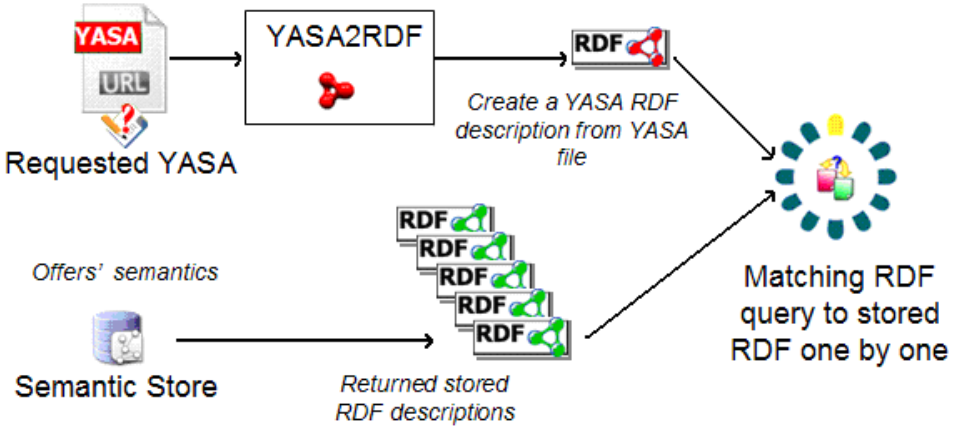


FIGURE 6.12 – L'appariement basé sur RDF.

Le principe d'appariement RDF est simple : une description RDF publiée est à retenir si et seulement si son graphe couvre le graphe de la description RDF de la requête. En effet :

1. Ceci peut être exprimé en SPARQL,
2. Ceci assure que pour chaque élément de la requête, il existe un élément dans l'offre annoté sémantiquement, donc que je peux comparer dans la deuxième phase jusqu'au dernier niveau d'appariement (potentiellement jusqu'aux inputs/outputs),
3. Ceci vérifie qu'il existe au moins autant d'opérations, d'inputs et d'outputs dans l'offre que dans la demande.

Je souligne ici que parfois certains nœuds ne sont pas annotés sémantiquement dans la requête ou dans l'offre. Pour assurer la conformité de la requête, nous avons conçu un générateur de requêtes SPARQL qui prend en entrée le graphe d'une interface d'un service Web requis et retourne une requête SPARQL correspondante. La Figure 6.13 représente un exemple d'une description YASAWSDL d'une requête et la Figure 6.14 représente son code SPARQL (de la requête).

```
1        <wsdl:interface name="Interface1">
2                <wsdl:operation name="Operation1"
3                        serviceConcept="Effect" modelReference="confirmation" />
4                <wsdl:operation name="Operation2"
5                        pattern="http://www.w3.org/ns/wsdl/in-out">
6                        <wsdl:output messageLabel="Output1"
7                                serviceConcept="Postcondition modelReference="checked""/>
8                </wsdl:operation>
9        </wsdl:interface>
```

FIGURE 6.13 – Description YASAWSDL d'une requête.

```
1        PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2        PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
3        SELECT ?service
4        WHERE {
5                ?service ?r1 ?interface .
6                        FILTER regex (xsd:string(?r1), "hasInterface") .
7                ?interface ?r2 ?op1 .
8                        FILTER regex (xsd:string(?r2), "hasInterfaceOperation") .
9                ?interface ?r3 ?op2 .
10                       FILTER regex (xsd:string(?r3), "hasInterfaceOperation") .
11               ?op2 ?r4 ?opt1 .
12                       FILTER regex (xsd:string(?r4), "hasOutput") .
13                       FILTER(!sameTerm(?op1,?op2))}
```

FIGURE 6.14 – Code SPARQL de la requête.

6.5 Conclusion

Dans ce chapitre, j'ai présenté l'annuaire des services Web sémantiques. J'ai repris en détails l'architecture et les rôles de ses composants. Ensuite, j'ai présenté les phases de publication et de découverte. Des détails concernant la réalisation de cet annuaire, les expérimentations de ses performances ainsi que son intégration au sein du projet national de recherche ANR SemEUsE [91] sont les sujets de la prochaine partie de ce manuscrit.

Chapitre 7

Réalisation

Sommaire

7.1 Introduction

La réalisation de nos contributions a été faite sur trois grandes étapes. Une première étape a été nécessaire pour concevoir et implémenter les algorithmes et les composants gérant la description, le stockage et la découverte des services Web sémantiques au sein de l'architecture de l'annuaire sémantique. Une deuxième étape a consisté à préparer un banc d'essai pour l'ensemble des contributions. Et la troisième étape évalue les performances de l'appariement et la découverte de l'annuaire sémantique de services afin de valider son intégration au sein du bus de services du projet ANR SemEUsE [1]. Ce chapitre ne reprend pas les détails techniques des développements du code source et de déploiement. Il est dédié plutôt à la présentation des démarches de réalisation des mécanismes et des processus pour mesurer les performances d'un système d'appariement et de découverte basé sur notre approche et d'en effectuer la mise au

1. SemEUsE : SemantiquE pour bUs de sErvices (http ://www.semeuse.org)

point et l'évaluation. Ceci nous a servi pour valider l'approche ainsi que de s'assurer que les performances soient meilleures que celles d'autres approches.

Dans ce chapitre, je présente la démarche de préparation d'un banc d'essai pour l'approche sémantique que je propose (Section 7.2). Le banc d'essai nécessite une génération de collections de test pour la description et la découverte (Section 7.2.1) et l'intégration du banc d'essai dans une plate-forme spécifique d'évaluations (Section 7.4.4). Puis, je présente les évaluations des performances de notre approche de découverte sémantique des services Web (Section 7.3). Ensuite, je résume un cas d'utilisation et le travail réalisé pendant le déploiement de notre approche de découverte dans le cadre de l'annuaire sémantique de services Web (Section 7.4). Enfin, je présente une conclusion pour ce chapitre (Section 7.5).

7.2 Préparation d'un banc d'essai de l'approche sémantique

Pour évaluer et comparer notre approche de découverte sémantique à des approches connues, nous avons développé des composants assurant description et appariement pour la découverte selon notre approche ensuite nous les avons intégrés dans une plate-forme d'évaluation spécifique aux services Web sémantiques. Les performances de nos mécanismes de découverte sont évaluées en exploitant des collections de tests à profil réaliste. Ils ont ainsi été validés et exploités par le bus sémantique de services.

7.2.1 Génération de collections de test pour la description et la découverte

Pour préparer le "banc d'essai" de mon approche sémantique, j'ai construit un générateur de descriptions et de requêtes de services Web. Ce générateur de descriptions permet de fournir des collections de tests à utiliser pour le "banc d'essai". Ainsi, dans le but d'obtenir une qualité de résultats la plus proche possible de la réalité, le générateur a été conçu afin qu'il produise des corpus de services qui respectent un profil fonctionnel et sémantique réaliste. Un profil fonctionnel et sémantique réaliste consiste à adopter une distribution particulière de paramètres fonctionnels et sémantiques dans un corpus de services générés. Cette distribution doit être la plus proche possible d'une distribution relevée sur un corpus de services effectifs qui sont plutôt construits un à un. Une distribution est définie par les occurrences des paramètres dans le corpus de services. Pour le profil réaliste, nous adoptons une approche reconnue dans le contexte du *Benchmarking* des services Web et présentée dans [74]. Les auteurs caractérisent un corpus de services Web à profil réaliste par la Figure 7.1. La ligne en noir foncé est la courbe réelle et celle en gris est son aspect qu'on arrive pas à voir vue l'échelle des valeurs.

L'axe des X de la Figure 7.1 indique le nombre d'occurrences dans le corpus de services d'un paramètre "*p*" donné. L'axe des Y indique le nombre de paramètres qui ont le même nombre d'occurrences (affiché sur l'axe des X). Dans le cas de notre description sémantique YASA, nous avons considéré que deux paramètres sont identiques s'ils portent le même couple d'annotations "*serviceConcept* + *modelReference*" et nous le nommons ici une "signature".

Notre objectif est de générer un corpus de services avec une distribution d'occurrences de signatures similaires à la distribution définie dans [74]. L'annotation des paramètres, les *inputs* et les *outputs*, utilise des concepts provenant de l'ontologie technique de services offerte par l'approche YASA, et des concepts d'une ontologie de domaine fourni avec la plate-forme d'évaluation. Nous

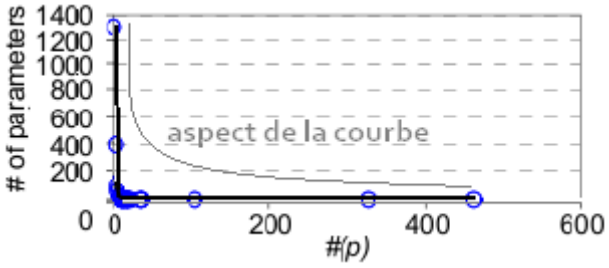


FIGURE 7.1 – Distributions de paramètres dans un corpus de services à profil réaliste.

avons sélectionné un ensemble de concepts de chaque ontologie. L'annotation des collections de tests est réalisée par un annotateur conçu spécifiquement pour respecter un profil sémantique réaliste. Le nombre de signatures qui ont un certain nombre d'occurrences a été établi avec la formule proposée dans [74]. La formule indique l'augmentation du nombre de signatures avec leur rareté (par exemple, quand une seule signature apparait 27 fois, dix signatures n'apparaissent que 3 fois).

Afin de préserver la forme réaliste de la distribution des signatures appliquées aux opérations et aux interfaces, nous définissons une distribution interpolée que nous leur appliquons. Cette distribution interpolée définit comme suit le nombre d'occurrences d'une signature :

$$\frac{(Nbr\ of\ occurrences\ in\ distribution) * (Nbr\ of\ element\ to\ annotate)}{Nbr\ of\ parameters}$$

C'est le nombre d'occurrences des paramètres dans la distribution multiplié par le nombre d'éléments à annoter (opérations ou interfaces), le tout divisé par le nombre de paramètres. Ensuite, pour chaque opération et interface la signature est tirée au hasard dans sa distribution interpolée correspondante et fourni par l'annotateur sémantique.

En ce qui concerne la génération des descriptions de services requis pour la découverte, et qui font partie des collections de test, chaque requête de découverte de services correspond à un service, choisi au hasard dans le corpus précédemment généré. Nous considérons qu'un service est équivalent à une requête et doit être retourné par les algorithmes d'appariement s'ils jugent qu'ils correspondent sémantiquement. Nous avons établi qu'un service est sémantiquement substituable à un autre, si leurs éléments syntaxiques sont annotés de la manière suivante :

- Pour les opérations, les interfaces, les outputs : les annotations de domaine du service offert doivent être soit des concepts identiques aux concepts de l'annotation domaine qui correspondent au même *serviceConcept* dans la requête, soit des sous-concepts des concepts de l'annotation domaine qui correspondent au même *serviceConcept* dans la requête.
- Pour les inputs : les annotations de domaine dans les services offerts doivent être des concepts identiques aux concepts de l'annotation domaine qui correspondent au même *serviceConcept* dans la requête, soit des concepts parents des concepts de l'annotation domaine qui correspondent au même *serviceConcept* dans la requête.

Pour effectuer les évaluations, nous avons réalisé un générateur de corpus de descriptions de services Web qui fourni des collections de test équivalentes dans les deux spécifications *YASA* et *SAWSDL*. Bien que les descriptions *YASA* soient plus riches sémantiquement (vu qu'elles

rajoutent la sémantique technique), le générateur respecte la spécification SAWSDL et apporte aux descriptions *SAWSDL* toute la sémantique fonctionnelle qu'elle peut supporter.

Pour obtenir des résultats encore plus proches de la réalité, chacun des corpus *YASA* et *SAWSDL* était constitué de quatre collections de tests avec différents nombres de requêtes et de services offerts pour la découverte. Elles sont notées *TC* pour "*Test Collection*". *TC1*, la plus petite collection, contient 2 requêtes pour 27 services candidats, *TC2* contient 20 requêtes pour 60 services candidats, *TC3* contient 45 requêtes pour 135 services candidats et finalement, *TC4*, la plus grande collection avec 90 requêtes pour 720 services candidats. L'ensemble des collections est nommé *YASA-TC* pour *YASA Test Collections*. La différence entre ces collections riside dans leurs tailles et leurs contenus.

La Figure 7.2 présente le profil de la distribution obtenue. L'axe des X indique le nombre d'occurrences dans le corpus de services générés d'un paramètre "*p*" donné. L'axe des Y indique le nombre de paramètres qui ont le même nombre d'occurrences (affiché sur l'axe des X). L'aspect de la courbe (en gris) montre que le corpus des descriptions de services et des requêtes offre un profil réaliste proche de l'aspect de la courbe de la Figure 7.1.

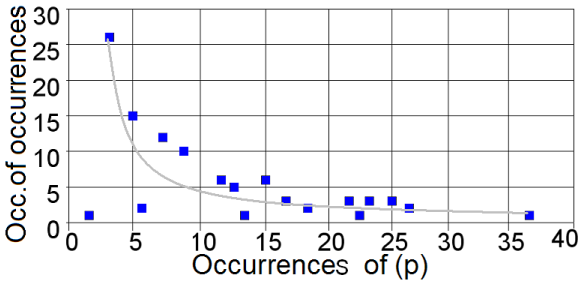


FIGURE 7.2 – Distributions de paramètres dans le corpus des collections de test

7.2.2 Intégration du banc d'essai de la découverte

Pour évaluer et comparer notre approche de découverte sémantique à des approches bien connues, nous avons développé et intégré nos composants d'appariement et de découverte sous forme de *plugins* adoptant nos approches dans la plate-forme d'évaluation de matchmakers sémantiques pour les services Web SME2 [84]. SME2 (pour *Semantic Web Service Matchmaker Evaluation Environment*) évalue les approches en se basant sur des bancs d'essai sur lesquels la plate-forme effectue des mesures de performances pendant des opérations de découverte. La version la plus récente est publiée en mai 2009[2], fourni des bancs d'essai pour les approches *OWL-S* et *SAWSDL*, ainsi que plusieurs fonctionnalités d'évaluation et d'exportation de résultats.

En fait, SME2 est le *Benchmark* du concours annuel reconnu "*Semantic Service Selection (S3)*", organisé depuis 2007 afin d'évaluer les matchmakers sémantiques. SME2 est utilisé pour :

- calculer les performances de découverte,
- évaluer les matchmakers des services Web sémantiques,
- réaliser des expérimentations sur des collections de tests publiques, appelées TC.

2. SME2 : http ://semwebcentral.org/projects/sme2/

Le Benchmark SME2 est utilisé pour l'évaluation de OWL-S matchmaker avec un corpus de services appelé OWLS-TC. Il est utilisé pour l'évaluation de SAWSDL matchmaker avec le corpus SAWSDL-TC. Ceci a permis à SME2 d'être au cœur du concours *S3*.

Pour participer, il faut :
- fournir une collection de tests contenant des services Web qui correspondent à la spécification de la description de services du matchmaker,
- implémenter l'algorithme dans un plug-in SME2 pour le matchmaker et fournir un fichier XML contenant des informations additionnelles,
- évaluer avec SME2 et tester les performances de découverte de l'algorithme sur la collection de test qui correspond au matchmaker.

Après avoir réalisé les corpus de descriptions, nous avons implémenté les trois variantes d'algorithmes d'appariement de l'approche des services Web sémantiques YASA, à savoir : l'algorithme Min-Average, l'algorithme Combinatory et l'algorithme Cupid. Chaque composant implémentant un algorithme est transformé en un plugin SME2 sous forme de fichier JAR. Un fichier XML contenant toutes les informations additionnelles nécessaires à la configuration du plugin, a été édité et associé à chaque variante.

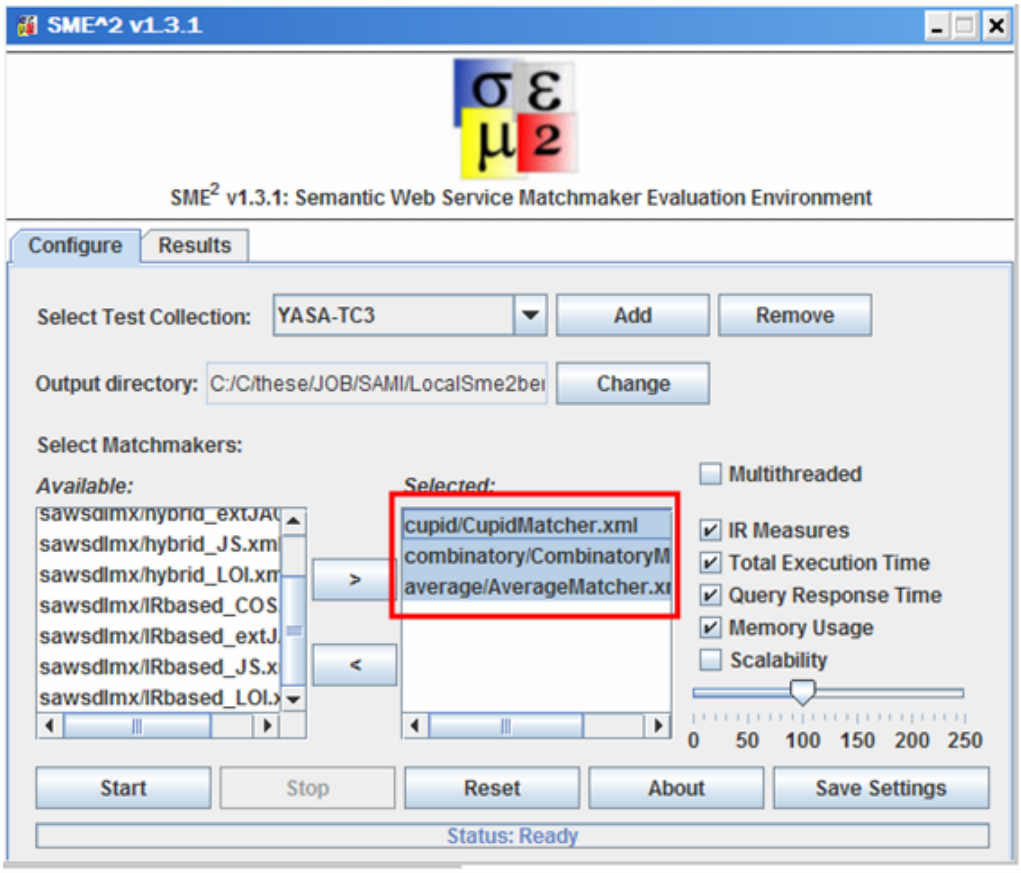


FIGURE 7.3 – Intégration des algorithmes d'appariement pour la découverte dans SME2

L'imprimé-écran de la Figure 7.3 montre que les plugins des algorithmes ont été détectés et bien intégrés dans le Benchmark SME2 (voir le cadre au milieu de la fenêtre). Ils peuvent être sélectionnés comme les autres plugins des matchmakers supportés par SME2. L'approche des services Web YASA peut être évaluée à travers les trois plugins Min-Average, Combinatory et

Cupid. La fenêtre de SME2 propose deux onglets Configure et Results. Dans le premier, on peut sélectionner : la collection à tester, le répertoire de sortie des résultats, les matchmakers, leurs plugins et les modes d'évaluations (IR, Memory, etc.)

7.3 Evaluations de la découverte sémantique de services Web

Les évaluations ont été réalisées en se basant sur des outils spécifiques aux évaluations des mécanismes de découverte et d'appariement offert par SME2. Pour être plus rigoureux dans l'évaluation, toutes les collections de tests ont été de profils réalistes et équivalentes sémantiquement à quelques différences exigées uniquement par la spécification de description, par exemple, l'ajout de la sémantique technique au niveau de l'attribut serviceConcept dans une description YASA par rapport à une description SAWSDL qui n'en possède pas.

7.3.1 Types d'évaluations

Pour valider nos propositions, nous avons fait deux types d'évaluations :

– Le premier type d'évaluation porte sur la comparaison des trois variantes d'appariement YASA-Matchmaker (YASA-M) de notre proposition de découverte,
– Le deuxième type d'évaluation se concentre sur la comparaison des trois variantes d'appariement YASA-Matchmaker de notre proposition de découverte, avec les variantes de l'appariement de la solution de découverte de *SAWSDL-MX*.

Pour évaluer les variantes (Min-Average, Combinatory et Cupid) de notre approche entre elles, nous avons procédé à des tests de découverte sur l'ensemble du corpus de services Web YASA. L'objectif de cette première évaluation était de noter le comportement de chaque variante de l'approche YASA-Matchmaker en augmentant "le nombre de requêtes" et "le nombre de services candidats" dans lesquels il existe des services pertinents pour chaque requête.

Pour évaluer les variantes (Min-Average, Combinatory et Cupid) de notre approche avec les variantes *SAWSDL-MX* (Logic, Cos, ExtJacc, JS et LOI), nous avons procédé à des tests de découverte sur les deux corpus YASA et SAWSDL réalisés pour le banc d'essai. L'objectif de cette deuxième évaluation était, d'abord, de noter le comportement de chaque approche globale, *YASA-M* et *SAWSDL-MX*, ensuite de chacune de leurs variantes par rapport à l'augmentation du "nombre de requêtes" et du "nombre de services candidats" dans lesquels il existe des services pertinents pour chaque requête, et qui sont équivalents de part et d'autre des spécifications YASA et SAWSDL.

Après avoir intégré les plugins des variantes d'algorithmes, nous avons ajouté à SME2 les collections de tests générées précédemment en YASA et SAWSDL. Une fois intégrées, les collections peuvent être sélectionnées pour être utilisées dans les évaluations. L'imprimé-écran de la Figure 7.4 montre comment sélectionner une collection de test (exemple, YASA-TC3 qui est encadré), ainsi que les catégories d'évaluations à effectuer (*IR Measures, Query Response Time,* etc.)

La Figure 7.5 montre un exemple de courbes et de figures fournies dans les rapports d'évaluations. La première partie de la figure présente le temps moyen de réponse aux requêtes et la seconde présente la consommation de mémoire. Cette figure donne un aperçu sur les rapports d'évaluation, les résultats sont présentées dans la section suivante.

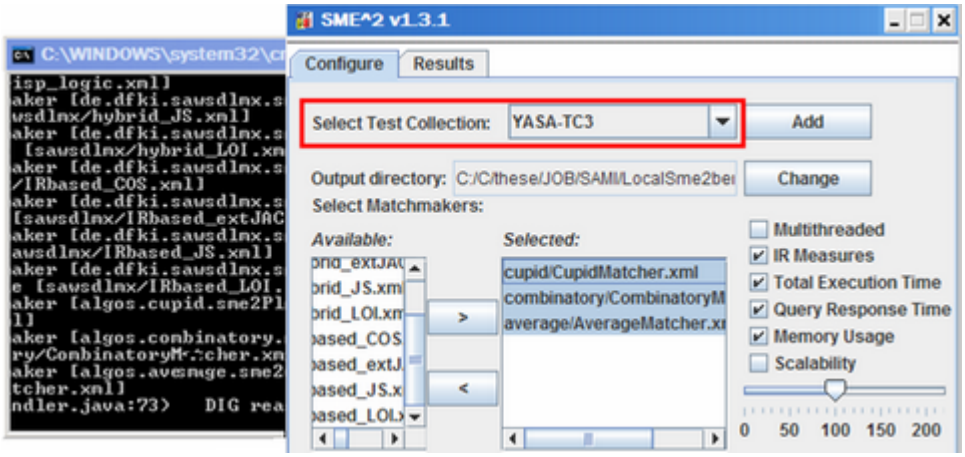


FIGURE 7.4 – Sélection d'une collection de test et lancement des évaluations

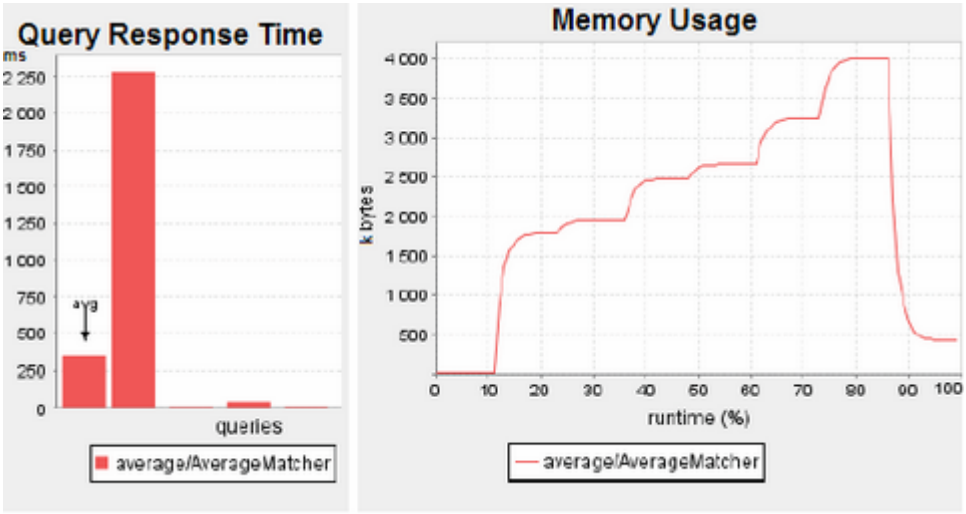


FIGURE 7.5 – Exemple des figures résultats des rapports d'évaluations

Dans la suite de cette section, je présente les résultats des évaluations basées sur le temps de réponse aux requêtes de découverte, le temps total d'exécution, la précision, le rappel et finalement le rapport rappel/précision.

7.3.2 Temps moyen de réponse aux requêtes

La première évaluation à présenter porte sur la comparaison des trois variantes d'appariement YASA-M de notre proposition de découverte. Nous évaluons ici les variantes entre elles en termes de temps moyen de réponse à l'ensemble des requêtes de découverte pour chaque collection de test.

En pratique, nous activons l'ensemble des plugins { *Min-Average , Combinatory , Cupid* }, nous choisissons l'évaluation nommée *Query Response Time*, nous sélectionnons la collection de test YASA-TC1 et finalement nous exécutons l'évaluation. Une fois l'essai fini, nous faisons la même démarche successivement avec les collections de test YASA-TC2, YASA-TC3 et YASA-TC4 une à une.

Pendant l'essai d'une collection de test, le temps de réponse à chaque requête est mesuré pour chaque variante YASA-M. Une moyenne du temps de réponse est calculée pour chaque variante YASA-M.

La Figure 7.6 présente le temps de réponse moyen aux requêtes des variantes (Min-Average, Combinatory et Cupid) de notre approche. La figure présente l'évaluation des performances de découverte pour les quatre collections de tests *TC1, TC2, TC3* et *TC4*. Les évaluations en termes de temps de réponse moyen aux requêtes rapportent que la variante Min-Average offre le plus rapide temps moyen de réponse aux requêtes de découverte lancées sur les quatre collections de test YASA de notre banc d'essai.

La valeur du temps moyen de réponse de Combinatory et Cupid augmente proportionnellement au nombre de services et de requêtes fournis dans les collections de test. En conclusion, Min-Average offre un meilleur comportement en termes de temps de réponse si le nombre de services et de requêtes augmente.

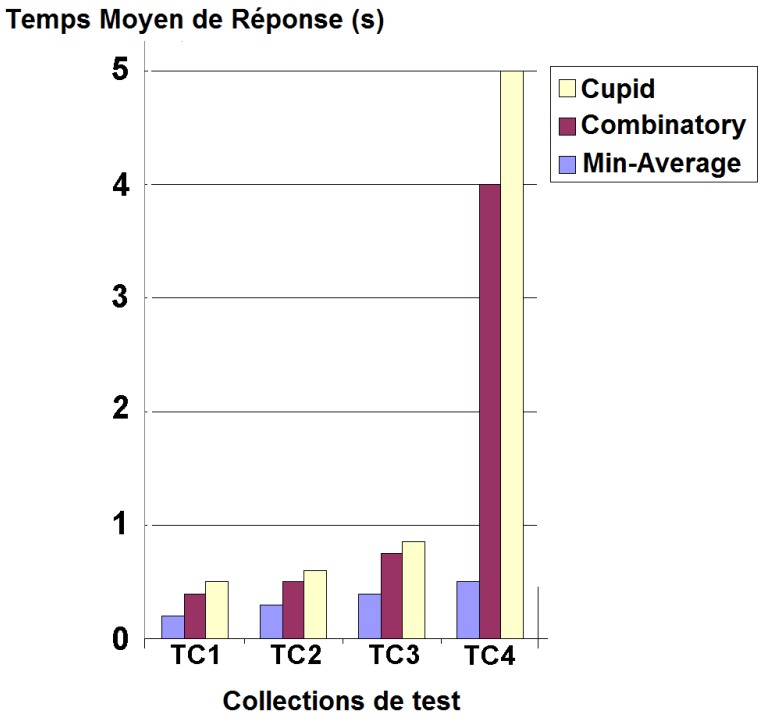


FIGURE 7.6 – Évaluations du temps de réponse moyen aux requêtes.

7.3.3 Temps d'exécution

La deuxième évaluation à présenter contient les deux types d'évaluations à savoir celle qui porte sur la comparaison des trois variantes d'appariement YASA-M de notre proposition de découverte et celle qui porte sur la comparaison des variantes d'appariement YASA-M et des variantes d'appariement SAWSDL-MX. Nous évaluons ici les variantes en termes de temps total d'exécution par rapport à l'ensemble des requêtes de découverte pour chaque collection de test.

Rappelons que nous avons réalisé un générateur de corpus de descriptions de services Web qui fourni des collections de test équivalentes pour les deux spécifications *YASA* et *SAWSDL*.

Donc notre banc d'essai offre des collections de test composées de deux groupes de services : YASA-TC et SAWSDL-TC sémantiquement équivalents.

YASA-TC est la collection de services décrites en YASA. SAWSDL-TC est la collection de services décrites en SAWSDL équivalente à celle de YASA-TC : on retrouve les mêmes noms de services, les mêmes opérations, les mêmes inputs et outputs et les mêmes annotations avec l'attribut modelReference. La différence entre les deux consiste à l'absence des annotations sémantiques avec l'attribut serviceConcept dans SAWSDL-TC.

En pratique, nous activons l'ensemble des plugins { *Min-Average , Combinatory , Cupid* }, nous choisissons l'évaluation nommée *Total Execution Time*, nous sélectionnons les collections de test YASA et finalement nous exécutons l'évaluation. Une fois le premier essai fini, nous activons l'ensemble des plugins { *Logic , LOI , JS , ExtJac , Cos* }, nous gardons la même évaluation mais nous sélectionnons maintenant les collections de test SAWSDL (équivalentes à YASA) et finalement nous exécutons l'évaluation.

Pendant l'essai d'une collection de test, le temps d'exécution de découverte pour chaque requête est mesuré pour chaque variante YASA-M et SAWSDL-MX. La somme des temps d'exécution est calculée pour chaque variante YASA-M et SAWSDL-MX.

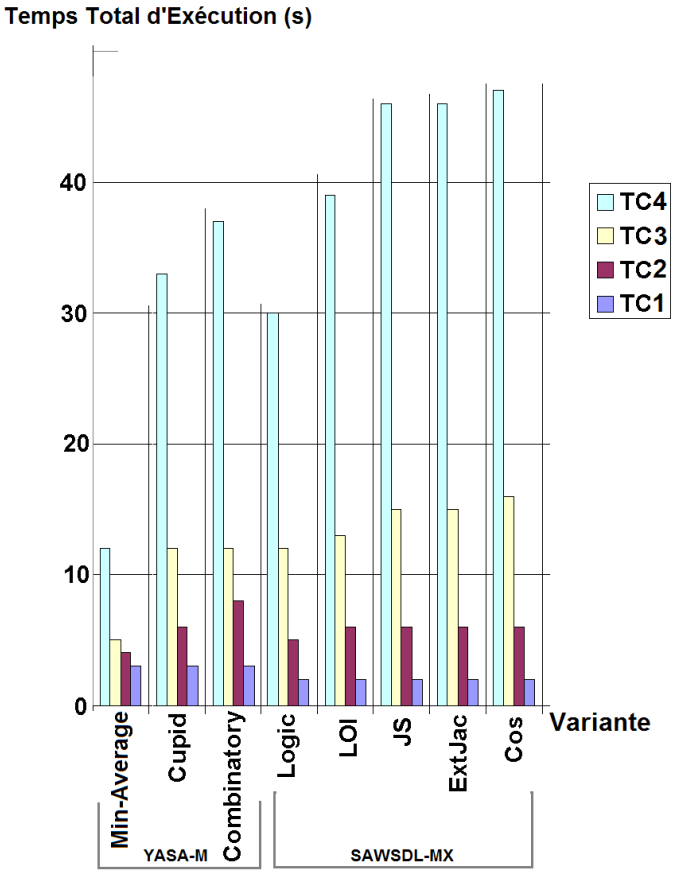


FIGURE 7.7 – Évaluations du temps d'exécution.

La Figure 7.7 présente le temps total d'exécution des variantes (Min-Average, Combinatory et Cupid) de notre approche avec les variantes *SAWSDL-MX* (Logic, Cos, ExtJacc, JS et LOI) calculé pour les quatre collections de tests *TC1, TC2, TC3* et *TC4*. Les évaluations en termes de

temps d'exécution démontrent que la variante Min-Average offre les plus rapides temps d'exécution.

Le temps d'exécution avec *SAWSDL-MX* et *YASA-M* augmente proportionnellement au nombre de services et de requêtes. Mis à part la variante *Logic* de *SAWSDL-MX*, toutes les autres variantes sont beaucoup moins performantes que les variantes *YASA-M* : Cupid et Combinatory réalisent des bonnes performances et sont classés parmi les meilleurs, juste après *Min-Average* (de *YASA-M*) et *Logic*.

En conclusion, Min-Average offre un meilleur rendement en termes de temps total d'exécution si le nombre de services et de requêtes augmente.

7.3.4 Précision

La troisième évaluation à présenter contient les deux types d'évaluations à savoir celle qui porte sur la comparaison des trois variantes d'appariement YASA-M de notre proposition de découverte et celle qui porte sur la comparaison des variantes d'appariement YASA-M et des variantes d'appariement SAWSDL-MX. Nous évaluons ici les variantes en termes de précision moyenne par rapport à l'ensemble des requêtes de découverte pour chaque collection de test. Nous utilisons dans cette évaluation les deux groupes de services de notre banc d'essai : YASA-TC et SAWSDL-TC.

En pratique, nous activons l'ensemble des plugins { *Min-Average , Combinatory , Cupid* }, nous choisissons l'évaluation nommée *IR Measures*, nous sélectionnons les collections de test YASA et finalement nous exécutons l'évaluation. Une fois le premier essai fini, nous activons l'ensemble des plugins { *Logic , LOI , JS , ExtJac , Cos* }, nous gardons la même évaluation mais nous sélectionnons maintenant les collections de test SAWSDL (équivalentes à YASA) et finalement nous exécutons l'évaluation.

Pendant l'essai d'une collection de test, la précision de découverte de chaque requête est mesurée pour chaque variante YASA-M et SAWSDL-MX. Une moyenne de la précision est calculée pour chaque variante YASA-M et SAWSDL-MX (voir Figure 7.8).

La Figure 7.8 présente la "valeur moyenne de précision" des variantes (Min-Average, Combinatory et Cupid) de notre approche avec les variantes *SAWSDL-MX* (*Logic* et *JS*) calculé pour les quatre collections de tests *TC1, TC2, TC3* et *TC4*. Les évaluations en termes de "valeur moyenne de précision" indiquent que toutes les variantes de *YASA-M* offrent une meilleure précision que la variante *Logic* et la variante *JS* de *SAWSDL-MX*.

La valeur de précision moyenne en *SAWSDL-MX* et *YASA-M* diminue proportionnellement au nombre de services et de requêtes. Min-Average offre une meilleure précision que la variantes Logique et JS de *SAWSDL-MX*. Min-Average assure aussi de très proches valeurs de précisions des autres variantes *YASA-M* (Combinatory et Cupid) quand le nombre de services et de requêtes augmente.

En conclusion, YASA-M offre un meilleur rendement en termes de précision par rapport à SAWSDL-MX si le nombre de services et de requêtes augmente.

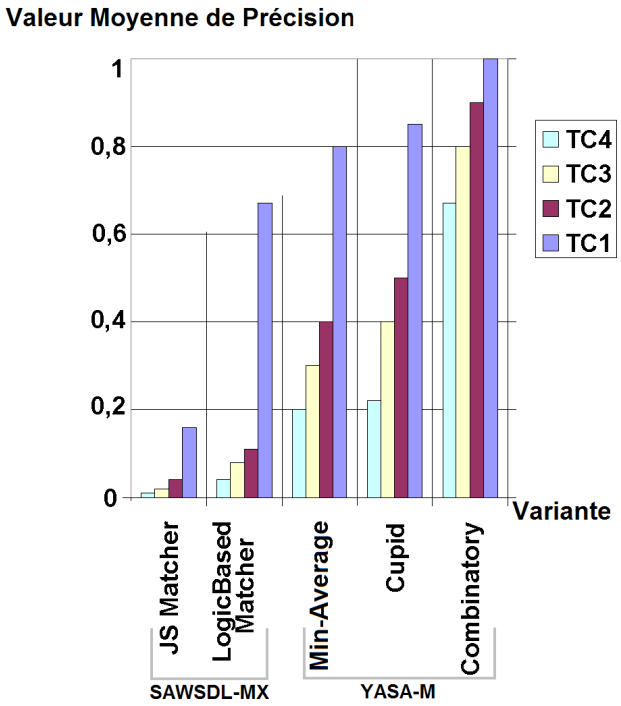


FIGURE 7.8 – Évaluations de la précision de la découverte.

7.3.5 Rappel

La quatrième évaluation à présenter contient les deux types d'évaluations à savoir celle qui porte sur la comparaison des trois variantes d'appariement YASA-M de notre proposition de découverte et celle qui porte sur la comparaison des variantes d'appariement YASA-M et des variantes d'appariement SAWSDL-MX. Nous évaluons ici les variantes en termes de valeur de rappel moyenne par rapport à l'ensemble des requêtes de découverte pour chaque collection de test. Nous utilisons dans cette évaluation les deux groupes de services de notre banc d'essai : YASA-TC et SAWSDL-TC (voir la Figure 7.9).

En pratique, nous activons l'ensemble des plugins { *Min-Average , Combinatory , Cupid* }, nous choisissons l'évaluation nommée *IR Measures*, nous sélectionnons les collections de test YASA et finalement nous exécutons l'évaluation. Une fois le premier essai fini, nous activons l'ensemble des plugins { *Logic , LOI , JS , ExtJac , Cos* }, nous gardons la même évaluation mais nous sélectionnons maintenant les collections de test SAWSDL (équivalentes à YASA) et finalement nous exécutons l'évaluation.

Pendant l'essai d'une collection de test, la valeur de rappel de découverte de chaque requête est mesurée pour chaque variante YASA-M et SAWSDL-MX. Une moyenne de la valeur de rappel est calculée pour chaque variante YASA-M et SAWSDL-MX.

La Figure 7.9 montre les "valeurs moyennes de rappel" pour chacune des variantes (Min-Average, Combinatory et Cupid) de notre approche et des variantes *SAWSDL-MX* (la variante *Logic* et la variante *JS*). Ces valeurs sont calculées pour les quatre collections de tests *TC1, TC2, TC3* et *TC4*. Les évaluations en termes de "valeur moyenne de rappel" précisent que toutes

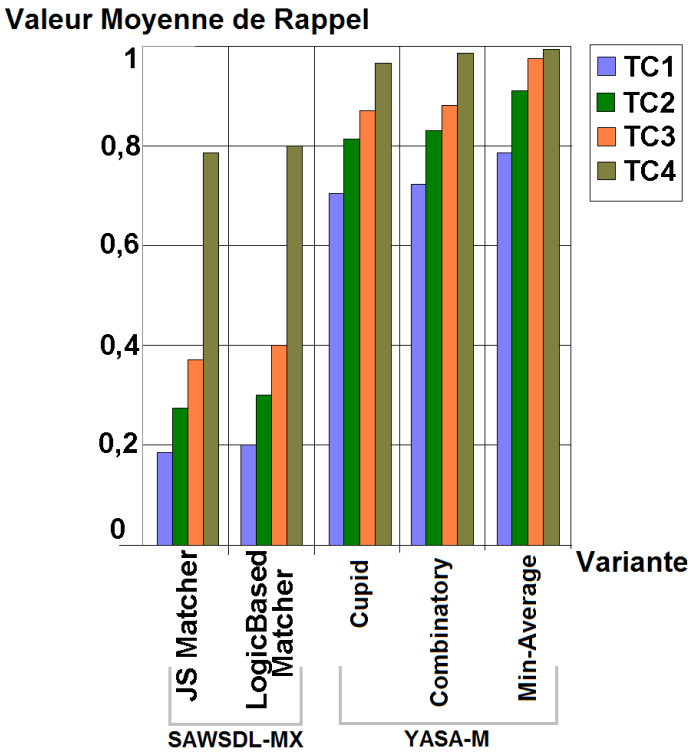


FIGURE 7.9 – Évaluations du rappel de la découverte.

les variantes de *YASA-M* offrent un meilleur rappel que la variante *Logic* et la variante *JS* de *SAWSDL-MX*.

La valeur moyenne de rappel en *SAWSDL-MX* et *YASA-M* varie proportionnellement au nombre de services et de requêtes. Min-Average offre un meilleur rappel que les variantes Combinatory et Cupid de *YASA-M*, ainsi que la variante basée sur l'approche logique et la variante JS de *SAWSDL-MX*, et offre de très proche valeurs de rappel des valeurs retournées par les variantes Combinatory et Cupid de *YASA-M* lorsque le nombre de requêtes et de services candidats augmente.

En conclusion, YASA-M offre un meilleur rendement en termes de rappel par rapport à SAWSDL-MX si le nombre de services et de requêtes augmente.

7.3.6 Rapport Rappel/Précision

En se basant sur les résultats des évaluations en termes de précision et les résultats des évaluations en termes de rappel, nous dressons un schéma illustrant le rapport de performance Rappel/Précision.

La Figure 7.10 montre le rapport "Rappel/Précision" pour chacune des variantes (Min-Average, Combinatory et Cupid) de notre approche et des variantes *SAWSDL-MX* (celle basée sur une approche logique et la variante JS). Ces valeurs sont calculées pour les quatre collections de tests *TC1, TC2, TC3* et *TC4*. La figure affiche en axe des X les valeurs de rappel et en axe des Y les valeurs de précision, de chaque variante. Exemple, le point de coordonnées proche de (0.2,0.2) qui correspond à la variante JS de SAWSDL-MX est construit à partir de la

valeur de rappel 0̃.2 de JS pour la TC1 sur la Figure 7.9 d'évaluation de rappel, et à partir de la valeur de précision 0̃.2 de JS pour TC1 sur la Figure 7.8. Le point suivant (0.3,0.1) de JS sur la Figure 7.10 correspond à la même chose mais pour TC2. Le point suivant (0̃.4,0.03) de JS sur la Figure 7.10 correspond à la même chose mais pour TC3. Le dernier point de la courbe de JS sur la Figure 7.10 correspond à la même chose mais pour TC4. De même pour les autres variantes.

Les courbes du rapport Rappel/Précision des variantes de chaque approche sont proches. Les meilleures courbes sont celles de l'approche de découverte offertes par les variantes de *YASA-M*. Bien que la courbe de la variante Combinatory soit légèrement meilleure que les autres, les variantes Min-Average et Cupid offrent aussi de très bonnes valeurs en termes de rapport Rappel/Précision qui restent proches de celles de la courbe de valeurs de la variante Combinatory.

Sur la Figure 7.10, on distingue bien la différence de performances de découverte des deux approches YASA et SAWSDL pour l'ensemble des collections de ce banc d'essai. YASA offre des variantes nettement plus performantes que celles de SAWSDL.

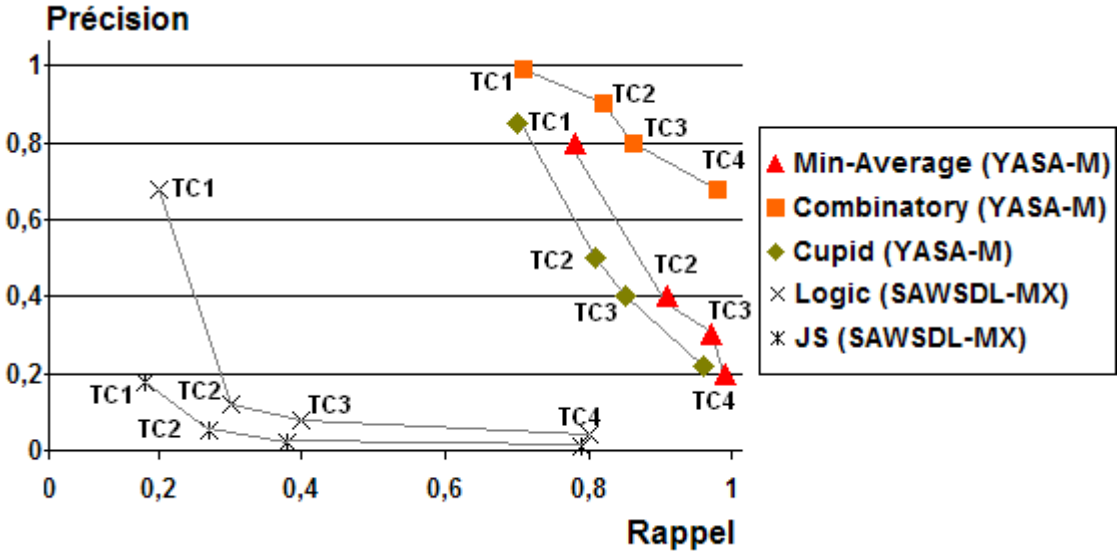


FIGURE 7.10 – Rapport Rappel/Précision.

7.4 Exploitation de la réalisation dans un cas d'utilisation

Dans cette section, je présente une vue globale des résultats du travail de développement réalisé afin d'implémenter nos contributions. Puis, je présente le processus de déploiement de l'annuaire sémantique des services Web. Ensuite, j'introduis brièvement le cas d'utilisation dans lequel notre approche de découverte sémantique de services Web a été exploitée. Enfin, je présente l'intégration de l'architecture de l'annuaire sémantique des services Web afin d'être utilisée dans l'étude de cas et par le bus de services sémantiques.

7.4.1 Développement de l'annuaire sémantique de services Web

Sans entrer dans les détails, je présente ici les principaux packages implémentant nos mécanismes de stockage et d'appariement, l'API développée pour utiliser l'annuaire et la construction du store sémantique. Ensuite, je présente la démarche de déploiement de notre approche de publication et découverte à travers l'annuaire sémantique SemeuseRegistry.

Principaux packages et classes

Tous les packages sont organisés et groupés selon leurs fonctionnalités dans les trois projets Java suivants :

- *SemeuseRegistry* : ce projet contient les interfaces et les classes de l'API de l'annuaire sémantique ainsi que celles en relation avec l'API de l'annuaire syntaxique,
- *SemeuseRegistryFunctionalColleage* : ce projet contient tous les packages qui traitent directement des composants fonctionnels de l'architecture de l'annuaire,
- *SemeuseRegistryUse* : Ce projet peut servir à la publication et à la découverte des fichiers WSDL des services Web sémantiques pour toute étude de cas.

Les principaux packages implémentant notre approche sont :
- *eu.itsudparis.semeuse.registry.api*
- *eu.itsudparis.semeuse.registry.impl*
- *eu.itsudparis.semeuse.registry.functional.discovery*
- *eu.itsudparis.semeuse.registry.functional.discovery.matching*
- *eu.itsudparis.semeuse.registry.functional.publication*
- *eu.itsudparis.semeuse.registry.functional.semanticstore*
- *eu.itsudparis.semeuse.registry.functional.discovery.semanticdb*
- *eu.itsudparis.semeuse.registry.api.dragonClient*

Comme leurs noms l'indiquent, ces packages concernent respectivement l'API de l'annuaire, l'implémentation de l'API, la découverte, l'appariement, la publication, le stockage sémantique, et enfin l'interaction avec la base sémantique et l'annuaire syntaxique *PetalsMaster* (anciennement nommé *Dragon*).

Le package "*eu.itsudparis.semeuse.registry.api*" contient des interfaces de l'annuaire, dont voici les principales d'entre elles :
- *DiscoveryMediator* : l'interface minimale que doit implémenter le médiateur de découverte,
- *PublicationMediator* : l'interface minimale que doit implémenter le médiateur de publication,
- *SemeuseRegistry* : l'interface de l'annuaire sémantique.

Le package "*eu.itsudparis.semeuse.registry.functional.discovery.matching*" contient des classes traitant l'appariement sémantique dont voici les principales d'entre elles :
- *SemanticMatching* : cette classe assure l'appariement élémentaire sémantique,
- *YasaMatching* : cette classe assure l'appariement sémantique global,
- *ResultObject* : cette classe retourne les résultats de la découverte ainsi que des éléments d'informations qui représentent une trace de l'appariement tels que les degrés d'appariement de chaque service avec la requête ainsi que la correspondance entre les

éléments de chaque service et ceux de la requête.

La Figure 7.1 présente l'API développée pour exploiter l'annuaire et exécuter la publication et la découverte.

TABLE 7.1 – API de l'annuaire sémantique.

void	**closeSession**() : calls the close session methods of the mediators.
DiscoveryResult[]	**discover**(**java.lang.String wsdlDescription, java.lang.String slaDescription)** discovers a set of services that matches with the given wsdl description.
java.lang.String	**getWSDL**(**java.lang.String dragonId)** It gets the complete WSDL description from the UDDI registry thanks to its dragonId and returns it as a string.
void	**openSession**() : calls the open session methods of the mediators, providing them the resources they require.
java.lang.String	**publish**(**java.lang.String wsdlDescription)** publishes the service of given URL sent as string.

Préparation du store sémantique RDF

Pour publier les données sémantiques des descriptions de services, nous avons besoin d'un registre RDF. Aujourd'hui, il existe deux implémentations *open source* de registre RDF : Jena [3] et Sesame [4]. Nous ne reprenons pas ici toutes les comparaisons que nous avons faites entre les deux implémentations mais nous donnons un résumé pour justifier le choix que nous faisons. Toutes les deux permettent le stockage de graphes RDF et d'ontologies, ainsi que l'exécution de requêtes sur les données stockées. Avec Jena, nous utilisons le langage d'interrogation SPARQL [5] qui est un standard W3C, alors que Sesame utilise SERQL qui n'est pas un standard (c'est uniquement en 2011 que Sesame 2 commence à supporter SPARQL [6]). En plus de cela, nous notons que pour inférer les propriétés des graphes RDF, Jena utilise un moteur d'inférence (*Reasoner*) dont l'implémentation est bien séparée de l'implémentation du store sémantique.

En résumé, le fait que Jena utilise le standard W3C SPARQL est très important pour nous. Un autre point que nous considérons est le fait qu'il est plus facile avec Jena qu'avec Sesame de changer de moteur d'inférence. Ceci est important à partir du moment que différents moteurs d'inférence peuvent offrir différents types d'inférences et qu'un moteur devrait être choisi selon les résultats attendus.

Une fois créé un store sémantique RDF qui est basé sur Jena, nous entamons la démarche de publication des descriptions de services dans le store. Pour cela, nous développons un *Parser* de descriptions sémantiques YASA. Le Parser doit stocker les descriptions dans des instances Java conformément au "modèle objet" de la spécification YASA. La Figure 7.11 résume comment nous préparons un *Parser* de descriptions sémantiques de services Web YASA.

3. Jena : A Semantic Web Framework for Java (http ://jena.sourceforge.net)
4. Sesame (http ://www.openrdf.org/sesame1.jsp)
5. SPARQL Query Language for RDF (http ://www.w3.org/TR/rdf-sparql-query/)
6. Sesame2 (http ://www.openrdf.org/download.jsp)

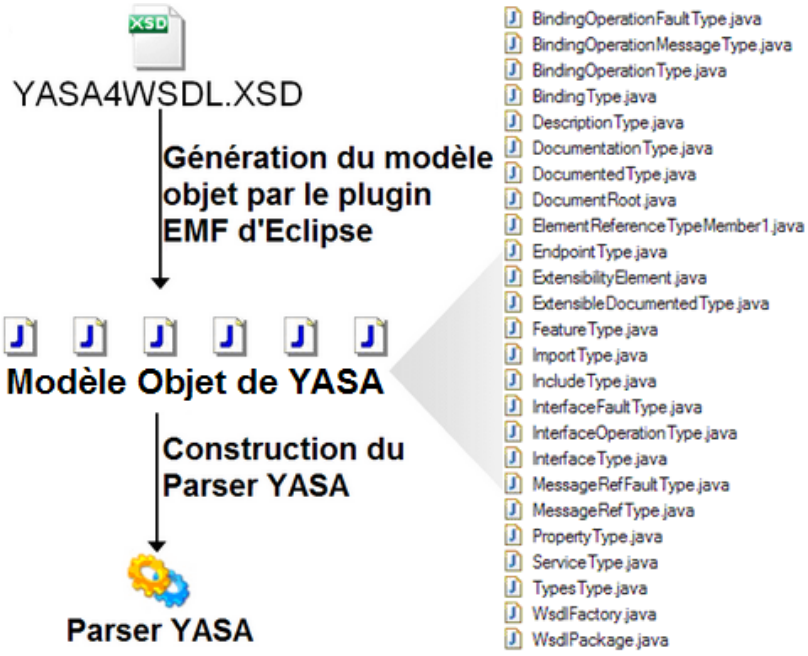


FIGURE 7.11 – Préparation d'un *Parser* de descriptions de services Web YASA.

Nous utilisons le plugin *EMF*[7] sur *Eclipse SOA*[8] pour générer les classes Java du modèle objet à partir du fichier de description *XSD* de la spécification YASA. Une fois les classes générées, le *Parser* est développé en les utilisant. Grâce à cela, il est capable de transformer une description WSDL annotée sémantiquement selon la spécification YASA en une instance d'objet Java en accord avec le modèle. Cette instance est traitée ultérieurement afin d'obtenir les parties sémantiques de la description du services Web. Notons que grâce à cette instance nous pouvons traiter autant les services Web YASAWSDL que SAWSDL, que ce soit en version WSDL 2.0 ou en version WSDL 1.1.

Pour la publication et la découverte, notre approche conçoit un store sémantique basé sur une base de données. La base de données est créée afin de correspondre aux spécifications de la description sémantique des services Web. Pour cela, nous avons créé une base de données MySQL[9]. La Figure 7.12 résume comment nous réalisons la publication grâce aux différents composants et outils développés.

En pratique, la publication se fait en trois étapes. Premièrement, notre *Parser* reçoit une description YASA et l'analyse. Si elle est une description valide (WSDL de YASA ou SAWSDL) alors il retourne un objet Java la représentant et en parallèle il renvoie la description WSDL telle qu'elle est à l'annuaire syntaxique PetalsMaster (ou Dragon). Deuxièmement, une fois la description publiée dans Dragon, ce dernier retourne son identifiant (ServiceID). Le contenu de l'objet Java et l'identifiant sont fusionnés et transformés en une description RDF. Troisièmement, la description RDF est stockée dans la base de donnée MySQL en utilisant Jena. La Table 7.2 présente le package et les classes implémentant la transformation YASA-RDF et le stockage des descriptions RDF dans le store sémantique.

7. EMF : Eclipse Modeling Framework Project (http ://www.eclipse.org/modeling/emf/)
8. Eclipse SOA (http ://www.eclipse.org/eclipsesoa/)
9. http ://www.mysql.fr

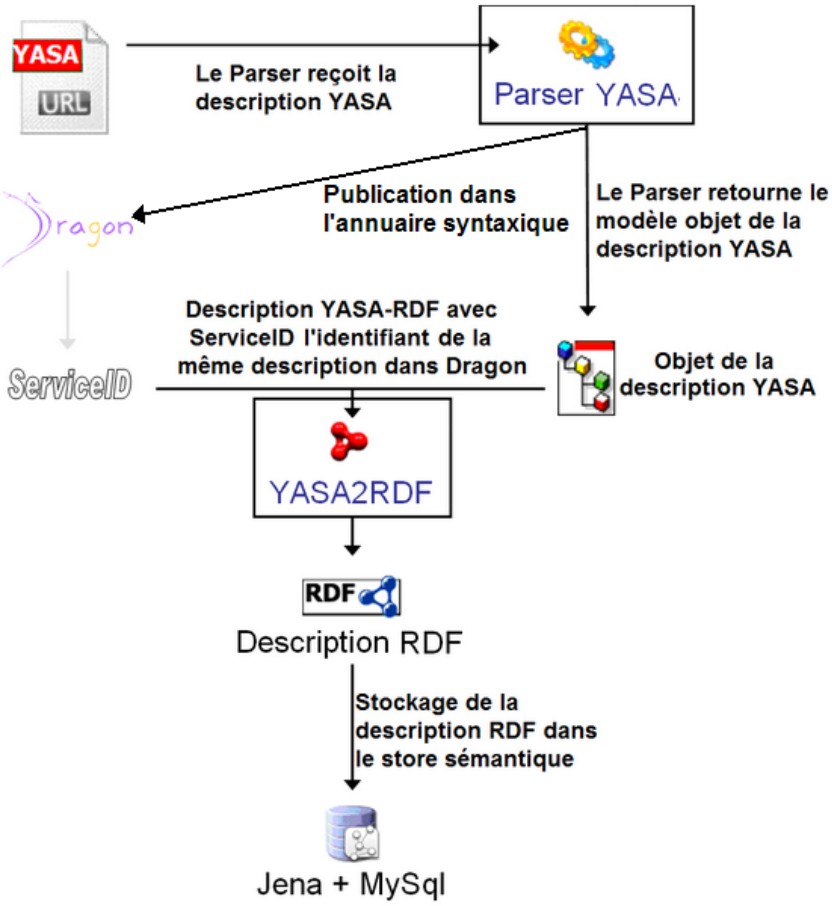


FIGURE 7.12 – Publication d'un fichier WSDL dans le store sémantique.

7.4.2 Déploiement de l'annuaire sémantique de services Web

Pour déployer notre approche de découverte sous forme d'un service Web, nous avons utilisé le serveur d'application Apache-Tomcat avec le package Apache Axis2. Étant plus modulaire et plus orienté XML que Axis, Axis2 nous a servi comme serveur SOAP et a fourni des outils pour le déploiement, le test et le monitoring des services Web de l'annuaire.

Toutes les classes implémentant nos mécanismes de publication, d'appariement et de découverte sont regroupées dans un répertoire contenant les projets Java composants la SemanticRegistry répertoriées sous "eu/itsudparis/semeuse/registry" et un dossier META-INF contenant un fichier *services.xml* qui configure le service Web de l'annuaire.

La Figure 7.13 présente le service Web de l'annuaire sémantique "*SemanticRegistry*" une fois déployé sous le serveur d'applications.

7.4.3 Cas d'utilisation : Collaboration pour résoudre une crise NRBC

Le projet national de recherche (SemEUsE [91]) a pour objectif de fournir une architecture de services Web sémantiques, sensible au contexte. Il est basé sur deux aspects : un premier aspect de conception, guidé par des modèles théoriques de services sémantiques et un deuxième

TABLE 7.2 – Package et classes implémentant la transformation YASA-RDF et le stockage.

Package eu.itsudparis.semeuse.registry.functional.yasa2rdf.translator	
SemanticDictionary	This class is a name factory, ensuring that the same rules are applied to the production of every name for the RDF descriptions elements.
YasaSemanticTranslator	This class is responsible for translating a java yasawsdl description into a set of RDF statements.
Package eu.itsudparis.semeuse.registry.functional.semanticstore	
YasaSemanticStore	This class is responsible for managing the Jena database and storing services semantical description in this database..

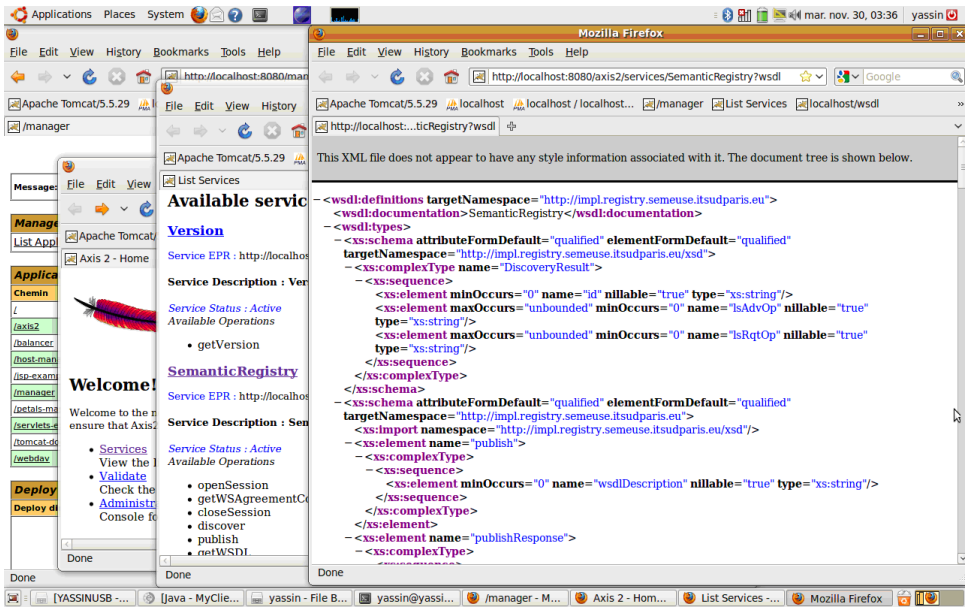


FIGURE 7.13 – Service Web de l'annuaire sémantique "*SemanticRegistry*".

aspect de *runtime*, guidé par un bus dynamique de services sémantiques (voir [91] pour plus de détails). Le projet met en œuvre ces approches à travers plusieurs études de cas. Nous avons participé à l'une de ces mises en œuvre en intégrant nos contributions et nos réalisations au sein du projet. Dans ce paragraphe, je donne brièvement une description du cas d'utilisation dans lequel notre approche de découverte sémantique de services Web a été exploitée afin de participer à la résolution d'un scénario d'une crise NRBC (Nucléaire / Radiologique / Bactériologique / Chimique).

D'après la définition du cas d'utilisation, "*dans une situation de crise (ex. catastrophe naturelle, explosion de violence), de nombreux intervenants, (ex. sécurité civile, policiers, fonctionnaires de santé, ONG) doivent agir simultanément et en urgence. L'interopérabilité est une composante majeure dans l'objectif de réduire la criticité de la situation. Ce problème s'inscrit dans une approche globale de sécurité et d'un besoin d'améliorer l'efficacité des réseaux d'acteurs impliqués dans un contexte critique*[10]."

10. SemEUse : *Use Cases Definition* (http ://www.semeuse.org/images/Docs/t5.1d1-use_cases_definitions.pdf)

Plusieurs acteurs peuvent intervenir dans de telles situations. Ils peuvent avoir plusieurs types, localisations et rôles. Les tâches des acteurs sont souvent représentées comme des services qui interagissent et collaborent pour résoudre les requêtes respectives des uns et des autres des acteurs. Ces services (invocation de tâches) doivent être soumis à des approches qui prennent en charge le traitement de risques, l'interopérabilité, la complexité et la systémique.

Le cas d'utilisation qui a été traité est la résolution d'une crise NRBC. Le contexte d'un exercice NRBC a comme référence un exercice fait par la Préfecture du Tarn [11] le 27 février 2004 [12]. Des Processus BPEL correspondant à ce contexte ont été établis et, enfin, la mise en œuvre de ce cas d'utilisation a été réalisée grâce une approche basée sur :

- la description sémantique de services Web selon la spécification YASA qui est utilisée pour décrire tous les services Web recherchés et invoqués par les intervenants dans la crise,
- la découverte et l'appariement sémantiques de services par *YASA-M*,
- l'orchestration de services en se basant sur les approches de QoS et de sécurité réalisées par les autres partenaires dans le projet SemEUsE.

Pour résumer, le scénario de l'étude de cas effectué pendant l'exercice simule une crise qui a lieu dans la station SNCF de Marssac (Tarn) à la suite d'une collision entre un camion-citerne (transportant un produit inconnu) et un wagon de fret contenant des produits chimiques. Notre approche doit assurer une description sémantique claire des différents services qui peuvent être offerts par les acteurs, ensuite elle doit assurer une découverte de services efficace et fiable sémantiquement afin de trouver l'ensemble de services requis. La séquence de services (représentés comme des tâches dans le processus BPMN) va être proposé aux acteurs afin qu'ils résolvent la crise NRBC.

Un BPEL de l'étude de cas NRBC a été établi. Il est traité par les composants d'orchestration du projet SemEUsE qui invoquent les différents services tout en suivant la démarche du processus décrit par le graphe BPMN.

La Figure 7.14 présente un extrait du grand graphe BPMN établi pour représenter le scénario de collaboration entre acteurs (services) dans la situation de crise NRBC.

La Table 7.3 résume les acteurs et leurs interventions sur le site de la crise. La Table précise pour chaque acteur son type, sa localisation et l'ensemble de ses rôles.

7.4.4 Intégration de l'annuaire sémantique dans l'architecture SemEUsE et exploitation pour l'étude de cas

Intégration de l'annuaire sémantique dans l'architecture SemEUsE

Pour valider nos réalisations, nous avons intégré l'annuaire sémantique à l'architecture de SemEUsE et exploité pour le cas d'étude précédemment présenté. D'une part, l'annuaire sémantique utilise en arrière plan l'annuaire syntaxique *PetalsMaster* (anciennement appelé Dragon). Des composants spécifiques ont été développés et ajoutés afin que les deux annuaires soient intégrés ensemble. D'autre part, l'annuaire sémantique est utilisé par les composants de SemEUsE qui traitent la QoS et le Contexte, et évidemment par toute autre composant du bus de services qui

11. Préfecture de Tarn : http ://www.tarn.pref.gouv.fr/
12. SemEUse : *Use Cases Definition* (http ://www.semeuse.org/images/Docs/t5.1d1-use_cases_definitions.pdf)

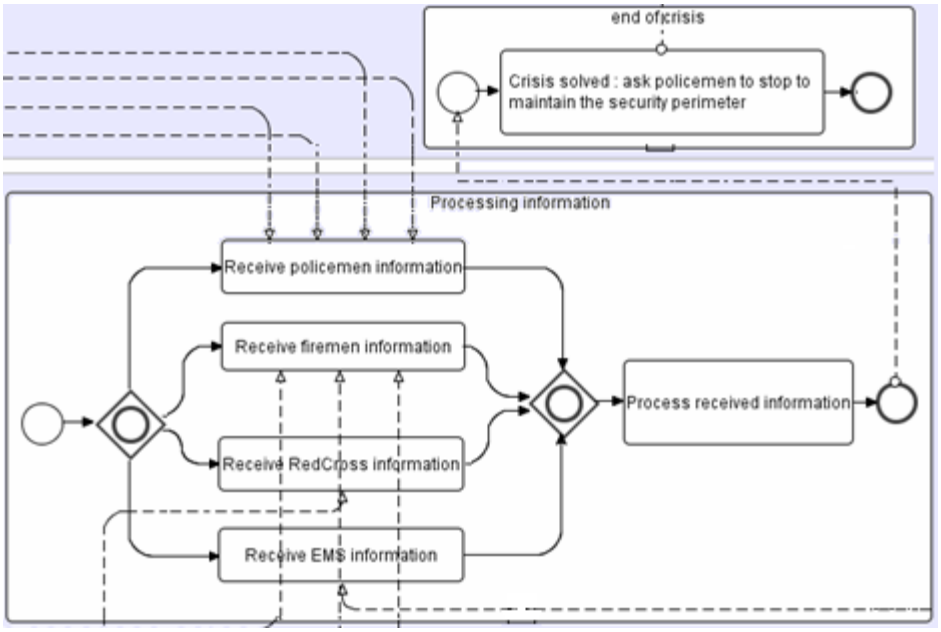


FIGURE 7.14 – Extrait du BPMN du cas d'utilisation : Situation de crise NRBC.

TABLE 7.3 – Les acteurs et leurs rôles dans le scénario de l'étude de cas.

Acteurs	**Types**	**Localisation**	**Rôles**
COD (Centre d'opérations de direction)	Humain	Préfecture	Gérer les actions pour la crise
Système d'information des Policiers	Humain	Distant (Internet, etc.)	Établir un périmètre de sécurité
Système d'information des Pompiers	Humain	Distant (Internet, etc.)	Combattre le feu et les explosions, chercher, soigner et ramener les victimes aux postes médicaux
Système d'information de La Croix Rouge	Humain	Distant (Internet, etc.)	Gérer les postes médicaux
Système d'information des services d'urgence (EMS)	Humain	Distant (Internet, etc.)	S'occuper des blessés

aurait besoin de publier ou de découvrir des services Web. Pour cela, nous avons collaboré avec nos partenaires dans le projet afin de connecter et tester tous ces composants ensemble avant d'exécuter le cas d'utilisation. Cette collaboration consistait principalement aux déploiements de clients de notre service Web d'annuaire chez les partenaires.

Exploitation de l'annuaire sémantique pour l'étude de cas

Par ailleurs, pour exécuter le cas d'utilisation, les services Web offerts par les acteurs intervenants dans la gestion de la crise ont été regroupés et mis à disposition afin d'être publiés, appariés et découverts pendant le scénario du cas de l'utilisation. Dans le cas d'utilisation, on doit publier dans l'annuaire sémantique quatre collections de services, chacune constituée de six instances de services Web sémantiques YASA. Les collections de services sont nommées : Military, DDE, Fireman et Samu et leurs services sont nommés : military_i, dde_i, fireman_i, samu_i ; avec $i \in [\,1\,,\,6\,]$.

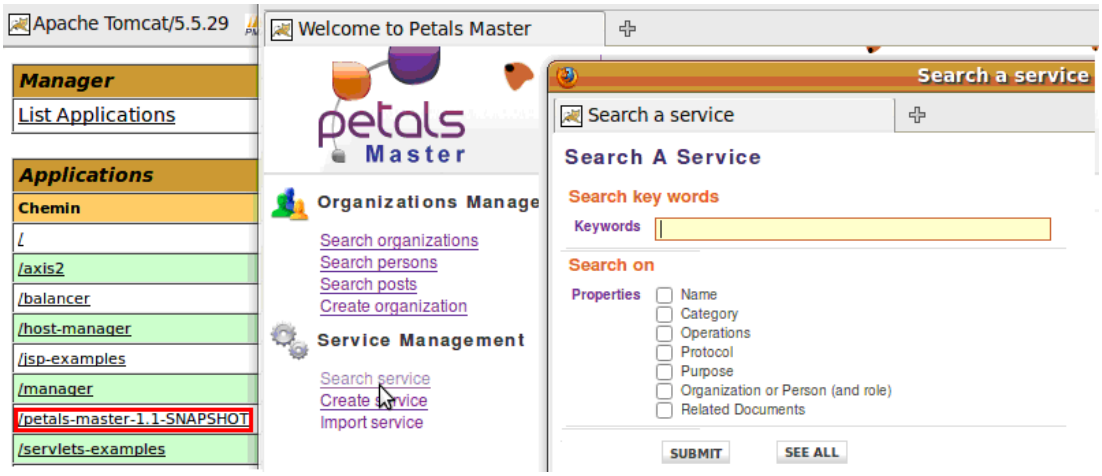


FIGURE 7.15 – Interface grahique de l'annuaire syntaxique *PetalsMaster*.

Au lancement du cas d'utilisation, les services Web décrits en YASA et offrant les services des acteurs sont publiés. Leurs contenus sémantiques sont analysés puis stockés dans le Store sémantique et leurs descriptions complètes sont publiées dans l'annuaire syntaxique.

Au lancement du scénario de gestion de crise, les services Web sont découverts en se basant sur les spécificités techniques et sémantiques précisées dans les descriptions des requêtes lancées à chaque fois que le processus a besoin d'exécuter une tâche. La Figure 7.15 présente l'interface graphique de l'annuaire syntaxique *PetalsMaster* une fois déployé sous le serveur d'applications. Dès qu'un service Web est publié par l'annuaire sémantique, il est publié dans *PetalsMaster*. Notre annuaire sémantique *SemeuseRegistry* est considéré désormais comme une extension sémantique de *PetalsMaster*.

La découverte ne se fait pas dans l'annuaire syntaxique, mais plutôt automatiquement à travers l'API de l'annuaire sémantique. Lorsque la découverte est exécutée, une requête de service Web par collection de services est lancée. L'annuaire de service effectue la découverte en se basant sur les annotations sémantiques de chaque service stockées dans le store sémantique. Un ensemble de services correspondant sémantiquement à chaque requête est retourné. Chaque liste de services est ordonnée par degré d'appariement avec la requête correspondante. Les listes sont renvoyées par l'annuaire aux autres composantes de SemEUse car nos travaux assurent un premier filtre des services découverts selon les exigences sémantiques et techniques des requêtes et nos partenaires sur le projet assurent des filtrages QoS, de Contexte et de Sécurité.

Pendant l'exécution du cas d'utilisation de la crise NRBC, tous les services qui sont censés correspondre aux requêtes dans le scénario, ont été découverts avec la pertinence attendue et les performances nécessaires pour le bon déroulement du scénario. Ce cas d'utilisation était utilisé lors de la démonstration du projet SemEUsE à l'ENSICA de Toulouse lors du colloque ANR SE2010. La Figure 7.16 présente une illustration récapitulative du projet SemEUsE. Elle présente la place de nos contributions en termes de description, stockage et découverte de services Web sémantiques.

Dans SemEUsE, plusieurs instances de notre annuaire sémantique sont créées et placées en réseaux distribués d'annuaires. Elles fournissent tous nos mécanismes d'appariement et supportent la spécification YASAWSDL ainsi que SAWSDL et bien sûr le standard WSDL.

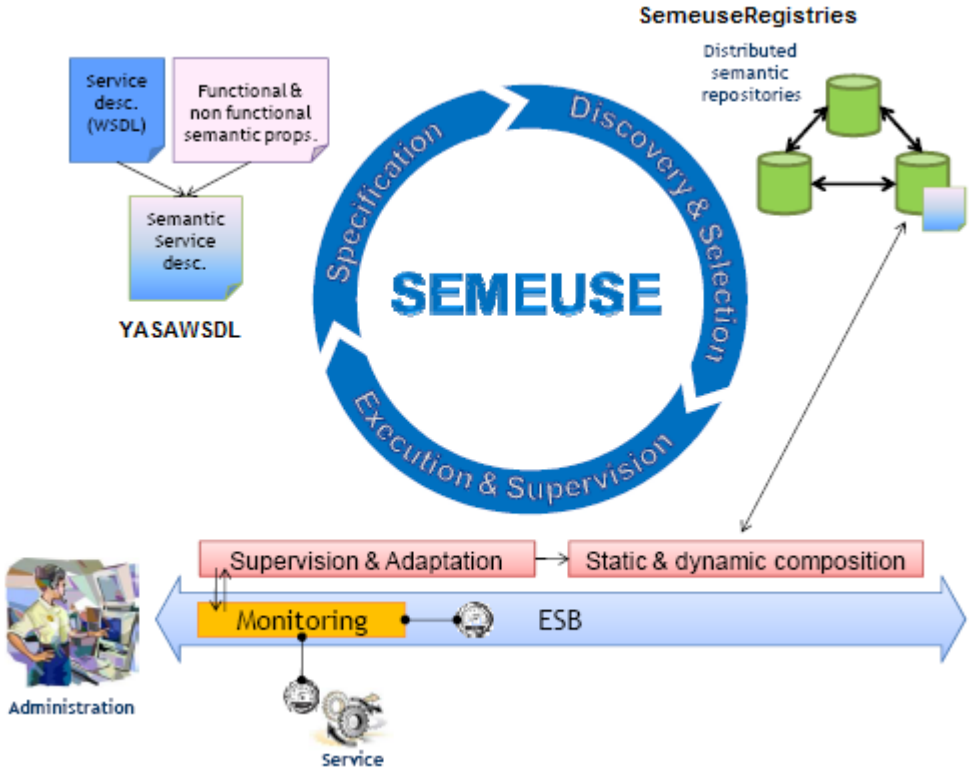


FIGURE 7.16 – Illustration récapitulative du projet SemEUsE.

7.5 Conclusion

Dans ce chapitre, j'ai présenté nos réalisations dans le cadre de nos travaux de recherche. J'ai commencé par la démarche de préparation d'un banc d'essai pour l'approche sémantique que je propose. Des collections de test pour la description et la découverte ont été générées et le banc d'essai a été intégré dans une plate-forme spécifique d'évaluations. Ensuite, j'ai présenté les évaluations des performances de notre approche de découverte sémantique des services Web. En fin, j'ai résumé le travail réalisé pendant le développement et le déploiement de notre approche de découverte dans le cadre de l'annuaire sémantique de services Web ainsi que l'exploitation de l'annuaire dans un cas d'utilisation du projet SemEUsE.

Quatrième partie

Conclusion générale

Conclusions et Perspectives

Bilan

Les travaux de recherche menés sur la description de services Web utilisent de plus en plus les ontologies pour fournir une représentation de l'information sémantique. Pour le stockage de services, les architectures d'annuaires de services Web n'offrent pas assez de composants dédiés à la prise en charge complète de la sémantique. En matière de découverte de services, les approches sémantiques actuelles présentent des lacunes. Les nombreux services disponibles aujourd'hui sur des annuaires purement syntaxiques ou supportant en partie la sémantique continuent d'exposer des problèmes d'imprécision et d'ambigüité par rapport à l'appariement sémantique, ce qui influe négativement sur la qualité d'une découverte. La découverte de services Web traditionnels consiste à apparier syntaxiquement les termes d'une requête à une description offerte.

L'étude de l'état de l'art nous a permis de mieux comprendre la problématique d'un point de vue système d'annotation, stockage et appariement sémantiques. Afin d'améliorer la description, le stockage ainsi que l'appariement et la découverte de services Web, il y a des manques à combler d'une part au niveau de la spécification de la nature de l'annotation sémantique dans certaines approches, et d'autre part, dans la prise en charge de la sémantique fonctionnelle dans les mécanismes de l'appariement et/ou dans le calcul des degrés d'appariements. Ces manques engendrent ambiguïté et imprécision lors de l'appariement et de la découverte.

Mes objectifs en termes de description de services Web sont de proposer une approche de description fonctionnelle explicite et précise tout en ayant la capacité d'ajouter des éléments non-fonctionnels et fonctionnels ultérieurement. La description proposée doit aussi s'adapter facilement aux standards et réutiliser les outils de services Web déjà disponibles. Mes objectifs en termes de découverte de services Web sont de proposer une approche d'appariement efficace avec des techniques d'appariement précises qui identifient précisément la similarité entre les éléments des deux services, requis et offert au moment de la découverte. Ceci doit être consolidé par une approche d'agrégation efficace des résultats d'appariements élémentaires afin de calculer avec précision le degré final/global d'appariement pour la découverte. Mes objectifs en termes de stockage sont de proposer une approche plus dirigée par la sémantique et l'automatisation des étapes de publication et de découverte.

Les résultats finaux consistent en un langage de description sémantique de services Web, un algorithme d'appariement et de découverte sémantiques de services Web, et finalement un annuaire de services Web sémantiques qui supporte la description de services et des composants de publication, d'appariement et de découverte basés sur les approches sus-citées.

Nos travaux de recherche s'intéressent au contexte d'architecture orientée services sémantiques, dans lequel les fournisseurs de services Web peuvent décrire, publier et découvrir des Services Web Sémantiques. Ces services sont accessibles et consultables en ligne depuis un annuaire sémantique. Mon approche vise à la proposition de techniques sémantiques pour rendre le processus de description et d'appariement de services efficace. Pour cela, les mécanismes sémantiques que nous proposons enrichissent la description par des concepts d'une ontologie technique provenant du domaine des Services Web. Ainsi notre approche de description précise-t-elle exactement la nature sémantique de chaque annotation (e.g. précondition, postcondition). En plus, l'approche continue à se baser sur les standards des services Web et s'inspire des travaux approuvés dans le domaine.

Dans cette thèse, j'ai présenté un langage de description sémantique de Services Web qui étend les recommandations et les standards existant (e.g. WSDL, SAWSDL, OWL-S) pour mieux décrire les informations sémantiques d'un service Web. La description de services Web que j'ai mis en œuvre dans cette thèse ne dépend pas d'un langage d'ontologie en particulier pour décrire l'aspect sémantique du service Web. Ma contribution en description de services Web est adaptée aux standards et réutilise les outils de services Web déjà disponibles. Cette description sémantique de services décrit la fonctionnalité avec le plus possibles d'éléments tout en ayant la capacité d'ajouter des éléments non-fonctionnels et fonctionnels ultérieurement. Mon langage de description est autant explicite que précis et couvre les éléments fonctionnels les plus importants. Avec le langage de description, j'ai présenté une ontologie technique de services pour l'annotation sémantique qui s'inspire des principales approches existants (e.g. OWL-S, WSMO).

J'ai présenté aussi une approche de découverte sémantique basée sur un algorithme d'appariement et d'agrégation sémantiques. L'algorithme d'appariement offre des techniques d'appariement qui parcourt la plupart des éléments des deux services, requis et offert, avec un mécanisme d'appariement qui identifie précisément et fidèlement la similarité entre les éléments des deux services, requis et offert. Des algorithmes d'agrégation ont été présentés. Ils calculent le degré final/global d'appariement pour la découverte. Un d'entre eux procède par l'arithmétique et les deux autres sont basés sur les graphes des descriptions de services Web.

Enfin, j'ai présenté un processus de publication et de découverte de services Web sémantiques au sein d'un annuaire sémantique de services Web. L'annuaire renforce la prise en charge de la sémantique avec des composants d'architecture bien dédiés à chaque aspect de stockage, d'appariement et de découverte. L'annuaire offre aussi une automatisation des phases de publication et de découverte qui avaient pour but l'amélioration des performances et la réduction au maximum de l'intervention d'utilisateur à la fois dans la "cataloguisation" des services au moment de la publication et dans la phase d'appariement pendant la découverte.

Ce travail a été validé d'une part en intégrant l'ensemble des composants proposés au sein du bus de SemEUsE. La partie de matchmaking (appariement) sémantique a été intégrée sur la plateforme d'évaluation SME2, ce qui nous a permis de montrer les bonnes performances tant qualitatives que quantitatives de nos propositions.

Perspectives

Les références des concepts dans les ontologies de services (de type ontologies techniques) doivent respecter une contrainte majeure que je rappelle ici. En effet, pour un élément donné WSDL (*service, interface, operation,* etc.) on ne peut pas référencer n'importe quel concept dans

une ontologie de services. Par exemple, un concept de comportement (qui peut être le concept "processus" dans une ontologie OWL service) ne peut être associé qu'aux éléments WSDL "*interface*" et "*operation*". Un concept de comportement ne peut pas être associé à un *"input"*. Par conséquent, une ontologie de services devrait être une extension du métamodèle de WSDL (des concepts de services Web WSDL comme *"service", "interface",* etc.) avec des concepts additionnels qui caractérisent les concepts représentant les éléments XML de la description WSDL (ex. *"operation", interface",* etc). Les concepts peuvent être reliés entre eux avec des relations de généralisation (*"is-a"*) mais surtout, ils sont reliés avec la relation *"hasProperty"*. Avec cette condition, l'attribut *"serviceConcept"* de YASAWSDL d'un élément WSDL "E" ne devrait référencer que des concepts qui sont liés au concept "E" dans l'ontologie technique de services. Dans YASAWSDL, l'annotation de la description du service, peut être faite "manuellement" ou à travers une interface graphique afin d'assister l'utilisateur. Il peut importer des ontologies et exposer leurs concepts afin de les utiliser dans les annotations. Tout élément descriptif ne pouvant être annoté par n'importe quel concept, un ensemble de contraintes d'annotation reste à définir et à implémenter avec l'interface d'annotation. Une des perspective de mon travail consiste à formaliser ces contraintes et à mettre en œuvre un outil d'aide à l'annotation des descriptions YASAWSDL.

Une deuxième perspective est liée à l'utilisation du mécanisme d'annotation sémantique YASAWSDL pour profiter des ontologies de contexte ou de QoS par exemple. En effet, j'ai proposé un langage de description sémantique de services Web qui étend SAWSDL et qui intègre la sémantique en se basant sur l'utilisation de deux types d'ontologies : un appelé "technique" qui offre des concepts de sémantique fonctionnelle (ex. : interface, effet, etc.) mais qui peut offrir aussi des concepts non fonctionnels (ex. : QoS, contexte, etc), et un autre type appelé "métier" ou "de domaine" qui offre des concepts de sémantique métier (ex. : tourisme, commerce, etc). Une perspective de mes travaux est de proposer en plus de l'ontologie technique fédératrice de services Web OTES une ontologie technique fédératrice de contexte ou de QoS pour les services Web sémantiques.

Une troisième perspective de mes travaux est le développement, la vérification et l'application de règles et de concepts additionnels dans l'ontologie technique de services Web pour permettre une composition efficace de services. En effet, en ajoutant de telles règles, la description sémantique des services Web en utilisant des annotations provenant de l'ontologie technique de services OTES peut devenir autant utile pour la composition que pour la découverte. Ces règles permettent de définir des liens et des relations entre des interface des services et/ou des opérations et surtout entre des *inputs/outputs*.

Une quatrième perspective est liée à l'hypothèse que j'ai faite pendant mes travaux et qui concerne l'annotation des services Web. Durant ce travail, j'ai supposé que c'est un utilisateur/expert qui fait l'annotation métier et technique. Une des perspective est d'assouplir cette hypothèse en offrant à l'utilisateur une annotation assistée. L'idée est d'assister l'utilisateur tout au long de la tâche d'annotation soit en lui proposant des concepts selon un ensemble de règles mettant en relation chaque élément XML d'une description WSDL (interface, operation, input et output) avec un sous ensemble de concepts provenant de l'ontologie technique. Cette démarche permet à la fois de (1) l'aider à choisir correctement l'annotation en évitant les annotations interdites par ces règles et (2) ultérieurement de lui proposer les annotations les plus fréquentes et utilisées grâce à un mécanisme d'analyse d'historique.

Bibliographie

[1] Frank Manola, Eric Miller. RDF Primer . http ://www.w3.org/TR/rdf-primer/, Février 2004.

[2] M. K. Smith, C. Welty, D. L. McGuinness. OWL Web Ontology Language Guide. http ://www.w3.org/TR/2004/RECowlguide20040210/, Février 2004.

[3] G. Alonso. Myths around web services. *Bulletin of the Technical Committee on Data Engineering*, 25(4) :3–9, Dec. 2002.

[4] A. Bansal, S. Kona, L. Simon, and T. D. Hite. A universal service-semantics description language. *Web Services, European Conference on*, 0 :214–225, 2005.

[5] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5) :34–43, 2001.

[6] A. Bernstein, E. Kaufmann, C. Kiefer, and C. Bürki. SimPack : A Generic Java Library for Similiarity Measures in Ontologies. Technical report, Department of Informatics, University of Zurich, 2005.

[7] A. Bernstein and C. Kiefer. iRDQL - Imprecise RDQL Queries Using Similarity Joins. In *3rd International Conference on Knowledge Capture (K-CAP)*, October 2005.

[8] A. Bernstein and C. Kiefer. Imprecise RDQL : towards generic retrieval in ontologies using similarity joins. In *SAC*, pages 1684–1689, 2006.

[9] P. Bertoli, A. Cimatti, and P. Traverso. Interleaving execution and planning for nondeterministic, partially observable domains. In *ECAI*, pages 657–661, 2004.

[10] Y. Chabeb and S. Tata. Yet Another Semantic Annotation for WSDL (YASA4WSDL). In *IADIS WWW/Internet 2008 Conference*, pages 462–467, October 2008.

[11] Y. Chabeb, S. Tata, and A. Ozanne. YASA-M : A Semantic Web Service Matchmaker. In *Proceedings of the 2010 24th IEEE International Conference on Advanced Information Networking and Applications AINA*, pages 966–973, 2010.

[12] J. Charlet, P. Laublet, and C. Reynaud. Web sémantique Rapport final. Rapport de recherche, Action spécifique 32 CNRS / STIC, FRANCE, Octobre 2003.

[13] R. Chinnici, J.-J. Moreau, C. A. Ryman, and S. Weerawarana. Web Services Description Language (WSDL) version 2.0 part 1 : Core language . http ://www.w3.org/TR/wsdl20/ (Avril 2011).

[14] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. http ://www.w3.org/TR/wsdl (Avril 2011), 2001.

[15] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (WSDL) Version 1.1. Technical report, W3C Note, Mars 2001.

[16] E. Cimpian, M. Moran, E. Oren, T. Vitvar, and M. Zaremba. Overview and Scope of WSMX. http ://www.wsmo.org/TR/d13/d13.0/v0.2/index.pdf (Avril 2011), February 2005.

[17] D. B. Claro, P. Albers, and J.-K. Hao. Approaches of web services composition - comparison between BPEL4WS and OWL-S. In *International Conference on Enterprise Information Systems (ICEIS 4)*, pages 208–213, 2005.

[18] DARPA. DARPA Agent Markup Language. http ://www.daml.org/about.html.

[19] De Bruijn J. et al. The web service modeling language WSML. WSML Final Draft, DERI, Octobre 2005.

[20] A. Elbyed. *Ontologie mapping pour l'intégration des données hétérogènes*. PhD thesis, TELECOM SudParis, 2008.

[21] ESSI WSMO Working Group. Web Service Modeling Ontology. http ://www.w3.org/Submission/WSMO/ (Avril 2011), 2005.

[22] D. C. Fallside and P. Walmsley. XML schema part 0 : Primer second edition . http ://www.w3.org/TR/2004/REC-xmlschema-0-20041028/ (Avril 2011), 2004.

[23] J. Fan, B. Ren, and L.-R. Xiong. An Approach to Web Service Discovery Based on the Semantics. In *FSKD (2)*, pages 1103–1106, 2005.

[24] J. Farrell and H. Lausen. Semantic annotations for WSDL and XML schema . http ://www.w3.org/TR/2007/REC-sawsdl-20070828/ (Avril 2011), 2007.

[25] D. Fensel and C. Bussler. The web service modeling framework WSMF. *Electronic Commerce Research and Applications*, 1(2) :113–137, 2002.

[26] B. Fuglede and F. Topsoe. Jensen-shannon divergence and hilbert space embedding. In *Information Theory, 2004. ISIT 2004. Proceedings. International Symposium on*, page 31, june 2004.

[27] E. Garcia. Cosine similarity and term weight tutorial, 2006.

[28] F. Giunchiglia, F. McNeill, and M. Yatskevich. Web service composition via semantic matching of interaction specifications. Technical Report DIT-06-080, University of Trento, 2006.

[29] D. Gusfield. *Algorithms on strings, trees, and sequences : computer science and computational biology*. Cambridge University Press, New York, NY, USA, 1997.

[30] B. He and K. C.-C. Chang. Statistical schema matching across web query interfaces. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 217–228, New York, NY, USA, 2003. ACM.

[31] Intelligent Data Exploration and Analysis Laboratory. Extended Jaccard Similarity . http ://www.lans.ece.utexas.edu/ strehl/diss/node56.html (Avril 2011), 2002.

[32] M. C. Jaeger, G. Rojec-Goldmann, G. Mühl, C. Liebetruth, and K. Geihs. Ranked Matching for Service Descriptions using OWL-S. pages 91–102, 2005. In Paul Müller, Reinhard Gotzhein, and Jens B. Schmitt, editors, Kommunikation in verteilten Systemen (KiVS 2005), Kaiserslautern, Germany, February 2005. Springer.

[33] H. Z. Jiwei, H. Zhu, J. Zhong, J. Li, and Y. Yu. An approach for semantic search by matching rdf graphs. In *Proc. of the Special Track on Semantic Web at the 15th International FLAIRS Conference (sponsored by AAAI*, 2002.

[34] K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28 :11–21, 1972.

[35] H. Kadima and V. Monfort. *Les Web Services*. Dunod, Paris, France, 2003.

[36] F. Kaufer and M. Klusch. WSMO-MX : A Logic Programming Based Hybrid Service Matchmaker. In *European Conference on Web Services*, pages 161–170, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[37] U. Keller, R. Lara, H. Lausen, A. Polleres, and D. Fensel. Automatic location of services. In *Proc. of the 2nd European Semantic Web Conference (ESWC)*, pages 1–16, Heraklion, Crete, 2005. LNCS 3532, Springer.

[38] C. Kiefer. *Non-Deductive Reasoning for the Semantic Web and Software Analysis*. PhD thesis, University of Zurich, Department of Informatics, Zürich, Switzerland, January 2009.

[39] C. Kiefer and A. Bernstein. The Creation and Evaluation of iSPARQL Strategies for Matchmaking. In *Proceedings of the 5th European Semantic Web Conference (ESWC)*, volume 5021 of *Lecture Notes in Computer Science*, pages 463–477. Springer-Verlag Berlin Heidelberg, 2008. to appear.

[40] C. Kiefer, A. Bernstein, H. J. Lee, M. Klein, and M. Stocker. Semantic process retrieval with iSPARQL. In *ESWC*, pages 609–623, 2007.

[41] M. Klein. Xml, rdf and relatives. *IEEE Intelligent Systems*, 16(2) :26–28, March/April 2001.

[42] M. Klein and B. König-ries. Coupled Signature and Specification Matching for Automatic Service Binding. In *Proc. of the European Conference on Web Services (ECOWS 2004*, pages 183–197. Springer, 2004.

[43] M. Klein and B. K"onig-Ries. Integrating preferences into service requests to automate service usage. In *First AKT Workshop on Semantic Web Services*, Milton Keynes, UK, Dezember 2004.

[44] M. Klein, B. Konig&#45 ;Ries, and M. Mussig. What is needed for semantic service descriptions : A proposal for suitable language constructs. *Int. J. Web Grid Serv.*, 1(3/4) :328–364, 2005.

[45] M. Klusch, B. Fries, and K. Sycara. Automated semantic web service discovery with OWLS-MX. In *Proc. of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 915–922, New York, NY, USA, 2006. ACM Press.

[46] M. Klusch and P. Kapahnke. Semantic Web Service Selection with SAWSDL-MX. In R. L. Hernandez, T. D. Noia, and I. Toma, editors, *SMRR*, volume 416 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[47] M. Klusch, P. Kapahnke, and F. Kaufer. Evaluation of WSML Service Retrieval with WSMO-MX. In *IEEE International Conference on Web Services, 2008. ICWS '08*, pages 401–408, Sept. 2008.

[48] S. Kona. Universal service-semantics description language. *Jonsson School Forum (JSF)*, 2006.

[49] D. Kourtesis and I. Paraskakis. Combining sawsdl, owl-dl and uddi for semantically enhanced web service discovery. In *Proceedings of the 5th European semantic web conference on The semantic web : research and applications*, ESWC'08, pages 614–628, Berlin, Heidelberg, 2008. Springer-Verlag.

[50] D. Kourtesis and I. Paraskakis. Web service discovery in the fusion semantic registry. In *BIS*, pages 285–296, 2008.

[51] D. Kourtesis, I. Paraskakis, A. Friesen, P. Gouvas, and A. Bouras. Web service discovery in a semantically extended uddi registry : The case of fusion. In L. Camarinha-Matos, H. Afsarmanesh, P. Novais, and C. Analide, editors, *Establishing The Foundation Of Collaborative Networks*, IFIP International Federation for Information Processing, pages 547–554. Springer Boston, 2007.

[52] U. Kuster and B. Konig-Ries. Semantic mediation between business partners - a sws-challenge solution using diane service descriptions. In *Proceedings of the 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Workshops*, WI-IATW '07, pages 139–143, Washington, DC, USA, 2007. IEEE Computer Society.

[53] U. Kuster and B. Konig-Ries. Semantic Service Discovery with DIANE Service Descriptions. In *WI-IATW '07 : Proc. of the 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Workshops*, pages 152–156, Washington, DC, USA, 2007. IEEE Computer Society.

[54] U. K"uster and B. K"onig-Ries. Supporting dynamics in service descriptions - the key to automatic service usage. In *Proceedings of the Fifth International Conference on Service Oriented Computing (ICSOC07)*, Vienna, Austria, September 2007.

[55] U. Küster and B. König-Ries. Evaluating semantic web service matchmaking effectiveness based on graded relevance. In *Proc. of the 2nd International Workshop SMR^2 on Service Matchmaking and Resource Retrieval in the Semantic Web at the 7th International Semantic Web Conference (ISWC08)*, Karlsruhe, Germany, October 2008.

[56] U. Küster, B. König-Ries, M. Klein, and M. Stern. DIANE - A Matchmaking-Centered Framework for Automated Service Discovery, Composition, Binding and Invocation. In *Proceedings of the 16th International World Wide Web Conference (WWW2007)*, Banff, Alberta, Canada, May 2007.

[57] R. Lara, D. Roman, A. Polleres, and D. Fensel. A conceptual comparison of WSMO and OWL-S. In *Proceedings of the European Conference on Web Services (ECOWS 2004)*, 11 2004.

[58] R. Lara, D. Roman, A. Polleres, and D. Fensel. A conceptual comparison of wsmo and owl-s. In L.-J. Zhang and M. Jeckle, editors, *Web Services : Proc. of the 2004 European Conference on Web Services*, volume 3250 of *Lecture Notes in Computer Science*, pages 254–269, Erfurt, Germany, September 2004. Springer-Verlag.

[59] Large Scale Distributed Information Systems. SAWSDL : Semantic Annotations for WSDL. http ://lsdis.cs.uga.edu/projects/meteor-s/SAWSDL/ (Avril 2011).

[60] H. Lausen and N. Steinmetz. Survey of current means to discover web services. Technical report, STI Innsbruck, 2008.

[61] C. Legner. Service-oriented computing - icsoc 2007 workshops. chapter Is There a Market for Web Services ?, pages 29–42. Springer-Verlag, Berlin, Heidelberg, 2009.

[62] V. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10 :707, 1966.

[63] V. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10 :707, 1966.

[64] Z. Liu, H. Wang, and B. Zhou. A scalable mechanism for semantic service discovery in multi-ontology environment. In *GPC*, pages 136–145, 2007.

[65] J. Madhavan, P. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *The VLDB Journal*, pages 49–58, 2001.

[66] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S : Semantic Markup for Web Services . http ://www.w3.org/Submission/OWL-S/ (Avril 2004).

[67] D. Martin, M. Paolucci, and M. Wagner. Bringing Semantic Annotations to Web Services : OWL-S from the SAWSDL Perspective. In *6th International and 2nd Asian Semantic Web Conference*, pages 337–350, November 2007.

[68] D. Martin, M. Paolucci, and M. Wagner. Toward Semantic Annotations of Web Services : OWL-S from the SAWSDL Perspective. In *OWL-S : Experiences and Directions Workshop, the 4th European Semantic Web Conference (ESWC 2007)*, The Tyrol region of Innsbruck, Austria, 2007.

[69] D. L. Martin, M. Paolucci, S. A. McIlraith, M. H. Burstein, D. V. McDermott, D. L. McGuinness, B. Parsia, T. R. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. P. Sycara. Bringing Semantics to Web Services : The OWL-S Approach. In *SWSWPC*, pages 26–42, 2004.

[70] Michael C. Jaeger. OWL-S Matcher. http ://owlsm.projects.semwebcentral.org (Avril 2011), 2005.

[71] A. E. Monge and C. P. Elkan. The field matching problem : Algorithms and applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 267–270, 1996.

[72] OASIS. ebXML Registry Services Specification v2.5 Committee Approved Specification. http ://www.oasis-open.org/committees/regrep/documents/2.5/specs/ebrs-2.5.pdf (Avril 2011).

[73] OASIS. UDDI. http ://xml.coverpages.org/uddi.html.

[74] S.-C. Oh, H. Kil, D. Lee, and S. R. T. Kumara. WSBen : A Web Services Discovery and Composition Benchmark. In *ICWS '06 : Proc. of the IEEE International Conference on Web Services*.

[75] OMG. Unified Modeling Language (UML) Version 1.5. http ://www.omg.org/technology/documents/formal/uml.htm.

[76] D. Orchard, A. S. Vedamuthu, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez, and Ümit Yalçinalp. WSDL 1.1 Element Identifiers. http ://www.w3.org/TR/2007/NOTE-wsdl11elementidentifiers-20070720/ (Avril 2011), 2007.

[77] OWL-S Coalition. Bringing semantics to web services : The OWL-S Approach. In *Semantic Web Services and Web Process Composition Workshop*, volume 3387 of *Lecture Notes in Computer Science*, pages 26–42, San Diego, CA, USA, July 2004.

[78] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic Matching of Web Services Capabilities. In *International Semantic Web Conference (ISWC)*, Sardinia, Italy, 2002.

[79] S. Potts and M. Kopack. *Sams Teach Yourself Web Services in 24 Hours*. Sams, Indianapolis, Indiana, USA, 2003.

[80] F. N. Sally Fuger and N. Stojanovic. ebxml registry information model version 3.0. oasis standard. May 2005.

[81] G. Salton. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.

[82] S. Schulte, M. Siebenhaar, and R. Steinmetz. Integrating semantic web services and matchmaking into ebxml registry. In *Proceedings of the Fourth International Workshop SMR2 2010 on Service Matchmaking and Resource Retrieval in the Semantic Web*, number 667, pages 69–83. CEUR Workshop Proceedings, Nov 2010.

[83] M. Schumacher, H. Helin, and H. Schuldt. *Semantic Web Service Coordination* . Chapter 4, CASCOM : Intelligent Service Coordination in the Semantic Web, Birkhäuser Basel, 2008.

[84] SemWebCentral. The Semantic Web Service Matchmaker Evaluation Environment . http ://www.semwebcentral.org/projects/sme2/.

[85] L. Simon, A. Bansal, A. Mallya, S. Kona, G. Gupta, and T. D. Hite. Towards a universal service description language. *Next Generation Web Services Practices, International Conference*, pages 175–180, 2005.

[86] K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller. Adding semantics to web services standards. In *International Conference on Web Services (ICWS'03)*, pages 395–401, Las Vegas, Nevada, June 2003.

[87] N. Srinivasan, M. Paolucci, and K. Sycara. Adding OWL-S to UDDI, implementation and throughput. In *In First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, pages 6–9, 2004.

[88] M. Stollberg, U. Keller, H. Lausen, and S. Heymans. Two-Phase Web Service Discovery Based on Rich Functional Descriptions. In *ESWC '07 : Proc. of the 4th European conference on The Semantic Web*, pages 99–113, Berlin, Heidelberg, 2007. Springer-Verlag.

[89] T. Andrews et al. Business Process Execution Language for Web Services, Version 1.1, Mai 2003.

[90] S. Tata, K. Klai, and N. O. A. M'Bareck. Coopflow : A bottom-up approach to workflow cooperation for short-term virtual enterprises. *IEEE T. Services Computing*, 1(4) :214–228, 2008.

[91] The French ANR SemEUsE Project. SemEUsE architecture . http ://www.semeuse.org/architecture.html.

[92] Université de Princeton. WordNet . http ://wordnet.princeton.edu/ (Avril 2011).

[93] W. E. Winkler and Y. Thibaudeau. An application of the fellegi-sunter model of record linkage to the 1990 u.s. decennial census. Technical Report Statistical Research Report Series RR91/09, U.S. Bureau of the Census, Washington, D.C., 1991.

[94] E. W. working group. WSDL Overview . http ://wsmo.org/TR/d24/d24.2/v0.1/.

[95] S. Zhu, J. Wu, and G. Xia. Top-k cosine similarity interesting pairs search. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2010 Seventh International Conference on*, volume 3, pages 1479 –1483, auguste 2010.