CBR4WSD : Une approche de découverte de services Web par Raisonnement à Partir de Cas

Ibrahim El Bitar

UNIVERSITE MOHAMMED V
ECOLE MOHAMMADIA D'INGENIEURS
RABAT

Centre d'Etudes Doctorales
Sciences et Techniques pour l'Ingénieur

Thèse de Doctorat

CBR4WSD : Une approche de découverte de services Web par Raisonnement à Partir de Cas

Présentée et soutenue publiquement le 13 Novembre 2014 par

Ibrahim EL BITAR

Pour obtenir le grade de

Docteur en Sciences et Techniques pour l'Ingénieur

Spécialité Informatique

Devant le jury composé de

Pr. Zohra BAKKOURY	**Président**	**EMI**
Pr. Fatima-Zahra BELOUADHA	**Directeur de thèse**	**EMI**
Pr. Ounsa ROUDIES	**Co-Directeur de thèse**	**EMI**
Pr. Khaled SMAILI	**Rapporteur**	**FS - UL**
Pr. Mounia FREDJ	**Rapporteur**	**ENSIAS**
Pr. Abdelilah MAACH	**Rapporteur**	**EMI**
Pr. Mahmoud NASSAR	**Examinateur**	**ENSIAS**

A mon vieux Liban demeurant toujours jeune et rayonnant,
Je suis fait de ton cèdre à jamais verdoyant et pétri de ta sainte glèbe,

Liban sans cesse en prospérité et débordant d'amour, je te suis à jamais,
Ô Liban des Phénix, plus que reconnaissant de te devoir le jour !

A mes chers parents,
Avec toute mon estime et affection.

Remerciements

Je voudrais exprimer mes sentiments les plus spontanés envers les personnes qui sans lesquelles ce travail de thèse n'aurait pas pu voir le jour. Leur aide, accompagnement et soutien m'ont été indispensables afin de pouvoir aboutir aux contributions de ma thèse.

Je voudrais tout d'abord exprimer ma reconnaissance envers tous les membres du jury pour la grande attention qu'ils ont bien voulue porter à mon travail.

Je suis très reconnaissant à mon co-directeur de thèse, Madame ***Ounsa ROUDIES****, Professeur de l'Enseignement Supérieur à l'Ecole Mohammadia d'Ingénieurs, pour tous les efforts administratifs qu'elle a fournis au service de mon dossier durant son parcours entre le ministère des Affaires Etrangères et de la Coopération Internationale marocain et l'ambassade du Royaume du Maroc à Beyrouth. Je la remercie aussi de m'avoir accueilli au sein de son équipe de recherche. Merci pour le temps et l'effort que vous avez consacrés à nos discussions.*

Je souhaite ensuite exprimer ma gratitude à mon directeur de thèse, Madame ***Fatima-Zahra BELOUADHA****, Professeur Habilité à l'Ecole Mohammadia d'Ingénieurs, qui m'a apporté le soutien scientifique et humain sans lesquels ces travaux de thèse n'auraient pas pu s'effectuer.*
Avec patience et pédagogie, elle m'a fait découvrir plusieurs facettes de l'activité de chercheur. Je lui dis ma reconnaissance pour l'aide compétente qu'elle m'a apportée, pour ses encouragements et pour la confiance qu'elle m'a toujours témoignée.
Je souhaite lui dire aussi que j'ai eu beaucoup de plaisir à travailler avec elle et que l'homme-chercheur que je suis aujourd'hui n'aurait pu être sans ses remarques et ses conseils judicieux.
Je la remercie aussi d'être ma grande sœur qui me soutenait en mes moments de lassitude afin de continuer de nouveau et réussir.

Je tiens à remercier vivement Madame ***Zohra BAKKOURY****, Professeur de l'Enseignement Supérieur à l'Ecole Mohammadia d'Ingénieurs, qui a accepté de présider le jury et d'examiner mon travail.*

Je remercie Monsieur ***Khaled SMAILI****, Professeur de l'Enseignement Supérieur à la Faculté des sciences de Beyrouth, Madame* ***Mounia FREDJ****, Professeur Habilité à l'Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes et Monsieur* ***Abdelilah MAACH****, Professeur Habilité à l'Ecole Mohammadia d'Ingénieurs, qui m'ont fait l'honneur de rapporter mon travail et pour leurs remarques enrichissantes.*

Je remercie aussi Monsieur ***Mahmoud NASSAR****, Professeur Habilité à l'Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes qui a accepté de faire partie de mon jury en tant qu'examinateur.*

Le métier d'enseignant-chercheur nous fait partager notre vie entre deux mondes.
J'aimerais aussi remercier les personnes qui font partie de « l'autre monde » :

L'équipe pédagogique et le personnel administratif du département Génie Informatique de l'Ecole Mohammadia d'Ingénieurs, pour ces trois belles années de monitorat passées en votre compagnie.
L'équipe pédagogique du département Génie Informatique de l'Ecole Nationale de l'Industrie Minérale, pour cette année d'enseignement agréable.

Mes remerciements vont aussi …

… à mes amis médecins:

Mohammad, Ghita, Abdel Kader, Abdel Rahman, Sondos, il m'est agréable de vous adresser mes remerciements pour tant de moments agréables partagés tout au long de mon séjour au Maroc.

… à ma famille EL ARJA:

Je remercie Dieu de m'avoir fait retrouver le foyer familial dont j'avais besoin, ici à Rabat, à 3988 *km du Liban, loin, très loin de mes parents et mon pays.*
Je remercie mes très chers EL ARJA pour leur générosité absolue et leur immense soutien. Nous avons partagé tant de bons moments. Grâce à vous tous, j'ai ainsi pu apprendre à aimer votre culture, votre musique, vos danses, votre langue, votre gastronomie, et votre sympathie!
Hajja Amina, Khaoula, Diaa, Hajar, Meriem et Youssef, je vous serai reconnaissant durant toute ma vie. Je vous aime beaucoup.

… à ma famille EL BITAR:

Un grand merci à ma famille pour son soutien tout au long de cette thèse.
Je remercie évidemment mes parents, Solod et Férial, pour leur irremplaçable et inconditionnel soutien et pour les sacrifices qu'ils ont faits en faveur de mon éducation et sans qui je n'aurai pas pu réaliser cette thèse. J'espère que le fruit de mes efforts leurs apportera fierté.
Enfin, je remercie ma chère sœur Natasha et mes chers frères Ammar et Hammoudi pour leur amour, leur soutien et leur encouragement constants.
Que Dieu vous bénisse.

Résumé

Le succès des services Web publiés dépend de la possibilité de les découvrir et par la suite les réutiliser. L'annuaire standard UDDI permet à l'utilisateur de rechercher des services Web publiés en fournissant un mécanisme de découverte fondé sur une recherche syntaxique. Toutefois, cette solution présente des limites dont l'importance croit avec l'évolution pléthorique des services Web.

Pour y remédier, de nombreux travaux ont proposé d'intégrer la sémantique dans la description des services. Le but est d'en assurer une meilleure interprétation et par la suite garantir une découverte efficace en améliorant la qualité des résultats obtenus. Ces travaux s'inscrivent dans trois catégories d'approches : l'approche algébrique, l'approche déductive et l'approche hybride qui vise à bénéficier des deux précédentes. Cette dernière comprend entre autres quelques travaux adoptant un raisonnement à partir de cas i.e. (Case Based Reasoning, CBR). L'objectif de ces travaux est de capitaliser sur l'expérience en permettant de réutiliser les solutions des cas similaires à un cas cible représentant une requête. Néanmoins, la plupart des travaux en question se limitent d'une part, à la découverte des services Web décrits selon l'ontologie OWL-S, et d'autre part, aux propriétés fonctionnelles des services.

Nous pensons alors qu'une approche de découverte de services Web alignée avec les standards W3C et couvrant différents aspects liés à cette problématique, mais aussi visant à optimiser le temps de découverte, demeure nécessaire pour améliorer la performance du processus de découverte.

L'approche CBR4WSD que nous proposons s'inscrit dans ce contexte. Elle constitue une nouvelle approche à base de raisonnement à partir des cas pour la découverte de services Web sémantiques. Son contour de contribution couvre un ensemble d'aspects qui visent à remédier aux limites des approches existantes. Ces aspects sont essentiellement relatifs à la rationalisation du traitement, à la maîtrise de la volumétrie des services Web traités, au respect des normes, et à l'amélioration de la qualité des résultats obtenus, en couvrant les besoins fonctionnels du client aussi bien que ses besoins non-fonctionnels. En vue d'atteindre ces objectifs, l'approche CBR4WSD utilise un modèle sémantique fondé sur les standards W3C. Ce modèle étend légèrement le standard WSDL et s'aligne avec les standards SAWSDL et WS-Policy. Il couvre les aspects fonctionnels et non-fonctionnels des services afin de permettre non seulement la découverte des services Web répondant à une requête mais aussi la sélection des meilleurs d'entre eux en prenant en considération des préférences du client. En outre, cette approche propose une organisation de la base de cas par communautés de services dans le but d'optimiser le temps de découverte. Dans cette même optique, elle utilise un processus de remémoration (recherche de cas similaires) qui se décline en un processus online et un autre offline. Ce dernier consiste principalement en un prétraitement réalisant l'appariement sémantique des concepts ontologiques mis en jeu, tandis que, le processus online se charge essentiellement du calcul dynamique des similarités entre les cas sources et cible. Finalement, l'approche proposée a été mise en œuvre par la mise en place du système CBR4WSD et validée à travers une expérimentation dans le domaine du tourisme, et en particulier, les services de réservation de billets d'avion.

Mots-clés : Services Web sémantique, Découverte de services, Raisonnement à Partir de Cas, Réutilisation, Conception par réutilisation, Appariement sémantique, WSDL, SAWSDL, WS-Policy.

Abstract

The success of the published Web services depends on the ability to discover and reuse them. The UDDI standard directory allows users to search for published Web services by providing a discovery mechanism based on syntactic research. However, this approach presents limitations.

To remedy this situation, many work have proposed to integrate semantics in the services description. The aim is to ensure a better interpretation and subsequently warrant an efficient discovery by improving the quality of results. These studies fall into three categories of approaches: the algebraic approach, the deductive approach and the hybrid approach. The latter one aims to fill the limits of the first two approaches in order to achieve an effective discovery. It includes some work adopting Case Based Reasoning. The objective of these work is to capitalize on the experience by reusing solutions of similar cases to solve a target case that represents a query. However, most of the concerned work are limited on the one hand, to the discovery of Web services described using OWL-S ontology, and on the other hand, to the services functional properties.

We believe that a Web services discovery approach aligned with the W3C standards and covering various aspects of this problem, but also aiming to optimize the discovery time, is still needed to improve the performance of the discovery process.

The CBR4WSD approach that we have proposed stands in this context. It is a new approach based on Case Based Reasoning for discovering semantic Web services. Its contribution outline covers a set of aspects which aim to overcome the limitations of existing approaches. These aspects are mainly related to the treatment rationalization, the control of the treated Web services volumetry, the compliance to standards and the improvement of the results' quality, by covering both clients' functional and non-functional needs. In order to achieve these objectives, the CBR4WSD approach uses a semantic model based on W3C standards. This model slightly extends the WSDL standard and aligns with the SAWSDL and WS-Policy standards. It covers the services functional and non-functional aspects to not only enable the discovery of Web services in response to a query but also to select the best services by taking into consideration the preferences of the client. In addition, this approach offers an organization of the case base by service community in order to optimize the time of discovery. Along these same lines, it uses a retrieval process (search for similar cases) that emanates in an online process and another offline one. The latter consists mainly on a pretreatment that performs the semantic matching of the involved ontological concepts. As for the online process, it is essentially charged of the dynamic calculation of similarities between source and target cases. Finally, the proposed approach has been implemented by the establishment of the CBR4WSD system and validated through experimentation in the field of tourism, particularly the flight booking services.

Keywords: Semantic Web Services, Services Discovery, Case Based Reasoning, Reuse, Design by reuse, Semantic Matchmaking, WSDL, SAWSDL, WS-Policy.

Liste des publications

- ## Publications dans des revues indexées

1. **Semantic Web Service Discovery Approaches: Overview and Limitations.**
   Ibrahim El Bitar, Fatima-Zahra Belouadha and Ounsa Roudiès.
   International Journal of Computer Science and Engineering Survey, ISSN: 0976-2760, Volume 5, Number 4, 2014.

2. **A CBR Based Approach for Web Service Automatic Discovery.**
   Ibrahim El Bitar, Fatima-Zahra Belouadha and Ounsa Roudiès.
   Journal Of Theoretical And Applied Information Technology (E-ISSN 1817-3195 / ISSN 1992-8645), 10 th April 2014, Vol. 62 No.1

3. **A Logic and Adaptive Approach for Efficient Diagnosis Systems using CBR.**
   Ibrahim El Bitar, Fatima-Zahra Belouadha and Ounsa Roudiès.
   International Journal of Computer Applications 39(15):1-5, February 2012. Published by Foundation of Computer Science, New York, USA. BibTeX (ISSN 0975 – 8887)

4. **Taxonomy and synthesis of Web services querying languages.**
   Ibrahim El Bitar, Fatima-Zahra Belouadha, Ounsa Roudiès
   International Journal of Science and Advanced Technology (ISSN 2221-8386), Volume 1 No 4 June 2011. http://www.ijsat.com

- ## Conférences internationales indexées avec publication d'actes

1. **Towards a semantic description model aligned with W3C standards for WS automatic discovery.**
   Ibrahim El Bitar, Fatima-Zahra Belouadha, Ounsa Roudiès.
   International Conference on Multimedia Computing and Systems April 14-16, 2014 Marrakesh, Morocco.

2. **Review of Web Services Description approaches.**
   Ibrahim El Bitar, Fatima-Zahra Belouadha, Ounsa Roudiès.
   8th International Conference on Intelligent Systems: Theories and Applications 08-09 May 2013, Rabat, Morocco.

3. **A Comparative Evaluation of Web Services Description Approaches.**
   Hajar Omrana, Ibrahim El Bitar, Fatima-Zahra. Belouadha, Ounsa Roudiès.
   10th International Conference on Information Technology: New Generations ITNG 2013, 15-17 April 2013, Las Vegas, Nevada, USA.

- ## Conférences nationales

1. **Etude et classification des langages de requêtes de services Web.**
   Ibrahim El Bitar, Fatima-Zahra Belouadha, Imane Rhalem, Ounsa Roudiès.
   1ère Journées Doctorales Informatiques JoDiC'12, Ecole Mohamadia d'Ingénieurs, 22-23 Février 2012, Rabat – Maroc.

2. **Approches de représentation des requêtes pour les services Web.**
   Ibrahim El Bitar, Fatima-Zahra Belouadha, Ounsa Roudiès.
   International Workshop on Information Technologies and Communication WOTIC'11, October 13-15 2011, Casablanca -Morocco.

3. **Solution aux imperfections de connaissances dans le RàPC.**
   Ibrahim El Bitar, Bilal Hussein, Fatima-Zahra Belouadha, Ounsa Roudiès.
   3ème Journées Doctorales en Technologie de l'Information et de la Communication JDTIC'11, 7-9 Juillet 2011, Tanger - Maroc.

Table des matières

Table des figures

Liste des tableaux

Introduction générale

SOMMAIRE

I. Contexte et problématique

Avec l'accroissement de la demande d'intégration dynamique d'applications d'entreprise, plusieurs architectures ont été proposées pour rendre les systèmes de l'entreprise adaptables à des environnements flexibles et évolutifs. L'architecture SOA (*Service-Oriented Architecture*), et en particulier son implémentation sous forme de services Web, constitue une solution de référence pour atteindre cet objectif.

Les services Web sont des composants logiciels interopérables fondés sur le standard XML, qui fournissent des fonctionnalités accessibles via des protocoles standardisés du Web. Leur large adoption comme technologie d'implémentation de l'Architecture Orientée Services (SOA) dans l'industrie logicielle est essentiellement justifiée par leur interopérabilité et leur standardisation. Notamment, l'interopérabilité est un critère crucial pour les applications effectuant des échanges indépendamment de leurs plateformes. En plus, l'élaboration d'une pile de standards de services Web contribue à la maturité de cette technologie.

Une activité d'ingénierie des services et des services Web par réutilisation s'est développée pour la mise en place de nouvelles applications ou la composition de nouveaux services répondant à des besoins complexes. Elle repose sur la découverte efficace de services Web pertinents et a suscité de nombreux travaux de recherche. Cependant, ce contexte demeure marqué par l'augmentation pléthorique du nombre de services Web sur Internet et par l'évolution des contraintes des clients, qui sont devenus plus exigeants et qui cherchent à réutiliser des services, répondant aussi bien à leurs besoins d'ordre fonctionnel (fonctionnalité attendue) que d'ordre non-fonctionnel (par exemple, des préférences en termes de qualité de service ou de sécurité). Vu ce constat, le besoin devient primordial d'élaborer de nouvelles solutions adoptant des mécanismes qui bénéficient des techniques du Web sémantique pour une découverte efficace et efficiente des services Web.

Largement utilisé, l'annuaire standard UDDI (Universal Definition and Description Integration) permet à l'utilisateur la sélection des services Web publiés répondant à sa requête, en fournissant un mécanisme de découverte fondée sur une recherche syntaxique par mots-clés. Toutefois, la solution offerte par l'UDDI est limitée par le fait qu'elle est incapable de découvrir des services dont les éléments de description ne sont pas syntaxiquement les mêmes que les mots clés spécifiés par l'utilisateur. Pour pallier cette limite, le concept de

services Web sémantiques a suscité l'intérêt des chercheurs dans ce domaine, notamment, les services Web décrits grâce aux ontologies. En effet, ces services sont, d'une part, aussi compréhensibles par les agents logiciels que par les utilisateurs humains, et d'autre part, ils permettent une interprétation correcte des informations envoyées et reçues dans le cadre de découverte, d'invocation et de composition [Berners-Lee et al., 2001]. Dans ce contexte, divers travaux de découverte de services Web ont été alors réalisés. Certains ont proposé des algorithmes d'appariement (matching) sémantique en utilisant particulièrement les ontologies WSMO (Web Service Modeling Ontology) ou OWL-S (Ontology Web Language for Service).

La prise en considération de l'aspect sémantique des services Web est certes nécessaire pour un processus de découverte de services efficace. Néanmoins, des mécanismes demeurent nécessaires pour garantir la sélection efficace de l'instance du service Web approprié, en termes de facteurs de qualité et de performance au moment de la consommation du service Web [Patil et Gopal, 2011]. Dans l'objectif d'améliorer les résultats de découverte, des approches permettant de sélectionner des services appropriés aux besoins du client ont été proposées. Elles consistent généralement à intégrer le contexte de l'utilisateur pour prendre également en considération des exigences non-fonctionnelles lors du processus de découverte [Sharma et Kumar, 2011].

Il est à noter que les travaux réalisés dans le domaine de découverte de services Web se répartissent en trois catégories d'approches : l'approche algébrique, l'approche déductive et l'approche hybride qui vise à combler les limites des deux premières approches. L'approche hybride comprend entre autres quelques travaux adoptant un raisonnement à partir de cas (RàPC) [Osman et al., 2006 a] [Osman et al., 2006 b] [Lajmi et al., 2006 a] [Lajmi et al., 2006 b] [Lajmi et al., 2007] [Wang et Cao, 2007] [De Franco Rosa et De Oliveira , 2008] [Osman et al., 2009] [Sun et al., 2009] [Lajmi et al., 2011] [Henni et al., 2013]. Le paradigme de RàPC ou *Case Based Reasoning (CBR)* issu de l'intelligence artificielle et de la psychologie cognitive est une approche fondée sur le retour d'expérience pour la résolution de problèmes. Cette approche s'appuie sur la réutilisation des cas déjà résolus pour chercher des solutions à des problèmes similaires. L'objectif des travaux de découverte ayant opté pour le RàPC est donc de capitaliser sur l'expérience, vu notamment le volume croissant des services Web disponibles qui est devenu un défi du processus de découverte. La plupart de ces travaux

ciblent, d'une part, les services Web décrits selon l'ontologie OWL-S définie principalement dans le domaine de la recherche, et d'autre part, ils se limitent aux propriétés fonctionnelles des services, ce qui réduit la pertinence des résultats de la découverte. Nous pensons alors qu'une approche de découverte de services Web alignée avec les standards W3C et couvrant différents aspects liés à cette problématique, mais aussi visant à optimiser le temps de découverte, demeure nécessaire pour améliorer la performance du processus de découverte.

C'est enfin dans ce contexte que s'inscrit cette thèse. Notre idée est d'exploiter le paradigme RàPC et les standards W3C pour la découverte des services Web, tout en proposant des mécanismes pour optimiser, d'une part, le temps de leur découverte, et d'autre part, améliorer la qualité des résultats, en permettant la découverte de services appropriés répondant aux besoins fonctionnels du client mais aussi à ses préférences et exigences particulières.

II. Objectifs de la thèse

Trois principaux objectifs ont guidé nos recherches :

- Proposer une approche de découverte de services Web sémantiques alignée avec les standards W3C de description fonctionnelle, sémantique et non-fonctionnelle des services Web. Les travaux connexes existants ciblent essentiellement des services décrits selon les modèles sémantiques OWL-S et WSMO définis dans des travaux de recherche. Or, en pratique, les services Web développés sont décrits conformément au standard du W3C WSDL.
- Exploiter le paradigme RàPC pour mutualiser les recherches antérieures et optimiser le temps de découverte. En effet, le succès des services Web a comme conséquence une croissance considérable du nombre de services publiés ainsi qu'une difficulté d'évaluer leurs aspects qualitatifs. Aussi, est-il nécessaire d'élaborer des mécanismes pour optimiser les temps de traitement et tenir compte des expériences réussies.
- Définir un mécanisme de découverte de services Web centré sur les besoins fonctionnels du client, mais qui intègre également ses besoins particuliers. Les travaux connexes optant pour l'approche RàPC se limitent généralement aux propriétés fonctionnelles des services Web ou les traitent au même niveau que celles non-fonctionnelles. Nous pensons que la sélection des services découverts doit non seulement cibler ceux qui répondent à la requête d'un client du point de vue fonctionnel, mais aussi ceux qui satisfont ses exigences, particulièrement, en termes de

qualité de service, et ce, tout en favorisant évidemment ceux qui assurent la fonctionnalité attendue.

III. Contributions

Nos propositions s'organisent tout naturellement autour des objectifs de recherche précédemment évoqués et viennent ainsi se cristalliser sous une méthode de découverte de services Web sémantiques fondée sur le RàPC, dénommée CBR4WSD : Case Based Reasonnig for Web Service Discovery. Son originalité réside dans son contour de contributions qui couvre un ensemble d'aspects visant à remédier aux limites des approches existantes. Ces aspects sont essentiellement relatifs à la rationalisation du traitement, le contrôle et la maîtrise de la volumétrie des services Web traités, le respect des normes, sans oublier l'amélioration de la qualité des résultats obtenus en termes de prise en considération de services capables de répondre aux besoins, aussi bien fonctionnels que non-fonctionnels, du client.

Comme l'a bien précisé Colette Roland, une méthode est un guide de développement d'un système complexe apportant, d'une part, un ensemble de modèles et de langages de représentation des produits de conception, et d'autre part, une démarche assistée par des outils pour aboutir à un produit de qualité [Roland, 2005]. Dans ce contexte, nos contributions s'articulent autour de ces quatre composantes.

1- Le modèle : Notre approche utilise un modèle de description sémantique des services Web fondé sur les standards W3C. Ce choix devrait permettre de l'appliquer aux services Web publiés qui sont en pratique décrits en WSDL. Le modèle élaboré étend légèrement le standard WSDL et s'aligne avec les standards SAWSDL et WS-Policy. Il introduit un attribut générique (ConceptType) dans le but de représenter l'aspect intentionnel d'un service Web (Goal) et le contexte de réalisation (préconditions et postconditions) Nous notons qu'il couvre en fait des aspects fonctionnels et non-fonctionnels des services, devant permettre de découvrir non seulement les services Web répondant à une requête, mais aussi d'en sélectionner les meilleurs en prenant en considération des préférences du client.

2- Le langage de représentation : Etant donné que notre approche est fondée sur le paradigme de RàPC, nous avons adopté une approche orientée problème/solution et identifié

l'ensemble des descripteurs nécessaires pour formaliser les cas manipulés dans notre système CBR4WSD. La requête de recherche d'un service Web a été également reformulée selon ce paradigme afin de disposer d'un formalisme unique des services Web manipulés, tout au long du processus de recherche. C'est dans cet esprit et conformément aux règles du RàPC que nous proposons notre modèle de représentation de cas qui est aligné avec les standards W3C.

3- La démarche : Elle décrit la façon d'organiser la production de notre système CBR4WSD. En effet, elle est composée de quatre étapes fondamentales : i) la formulation de la requête de service Web via un Template conforme au modèle de description sémantique proposé, ii) l'élaboration des cas cibles dans l'esprit du RàPC, iii) la recherche des cas similaires au cas cible (la requête) dans la base de cas et iv) la proposition d'un ensemble de cas qui répondent à la requête initiale.

Dans un souci d'optimisation de la découverte, nous avons proposé plusieurs mécanismes qui consistent en :

- l'organisation de la base de cas dans le but de rationaliser le traitement et d'optimiser le temps de découverte. Nous optons pour l'organisation par communauté de services en fonction de l'aspect intentionnel (goal).
- un processus de remémoration (recherche de cas similaires) fondé sur un appariement sémantique et qui se décline en deux processus : un processus online et un processus offline. Le processus online couvre cinq couches de traitement : la formalisation, la découverte, la sélection, le test et la mémorisation. Quant au processus offline, il correspond à une couche transversale assurant l'administration et la configuration du système de découverte.
- une sélection pertinente effectuée en deux temps : la sélection fonctionnelle qui est prioritaire est filtrée plus tard selon les critères non-fonctionnels tels que la qualité de services.

4- L'outil d'assistance : L'objectif est double : faciliter la recherche dans la base de cas et expérimenter notre démarche. En effet, nous avons mis en œuvre le système CBR4WSD par l'élaboration et l'implantation d'un ensemble d'algorithmes relatifs aux processus de remémoration. Au niveau de la remémoration offline, ces algorithmes sont destinés à la gestion de la base de cas et de la base de communautés de services, aussi bien qu'à

l'appariement sémantique inter-concepts. Nous avons aussi implanté les algorithmes traduisant les principes du processus de remémoration online.

IV. Organisation du mémoire

La suite de ce manuscrit est structurée selon trois parties. La première partie expose le contexte global de manière à positionner notre contribution et à identifier les défis à relever. La deuxième partie présente les différents volets de notre contribution CBR4WSD tandis que la troisième est consacrée à son évaluation.

La première partie est composée des chapitres 1, 2 et 3.
Le **premier chapitre** introduit les principes du paradigme de Raisonnement à Partir des Cas. Il présente le carré d'analogie dirigeant le RàPC et décrit son cycle de référence. Quelques techniques de recherche de similarité y sont également exposées.
Le **second chapitre** présente une analyse comparative qui explore les approches de description des services Web, à savoir les standards W3C (WSDL 2.0, SAWSDL et WS-Policy 1.5) et les deux modèles sémantiques OWL-S et WSMO.
Dans le **troisième chapitre**, nous analysons des approches de découverte de services Web en vue d'identifier les mécanismes qui font leurs points forts, et qui par la suite, pourront servir de guide pour déterminer les fondements d'une approche de découverte optimisée. Un intérêt particulier est apporté aux approches optant pour le RàPC, à l'instar de nos recherches.

Quant à la deuxième partie, elle décrit CBR4WSD, notre méthode de découverte de services Web sémantiques par le RàPC. Elle est organisée en quatre chapitres (4 à 7) correspondant à ses composantes i.e. les modèles, le langage de représentation, la démarche et l'outil.
Le **quatrième chapitre** décrit les fondements de notre approche de découverte de services Web. Nous commençons par élucider l'ensemble des critères retenus qui délimitent le contour de notre contribution. Nous y exposons ensuite les mécanismes et les fondements de cette approche CBR4WSD et le processus central de recherche.
Le **cinquième chapitre** se focalise sur la phase d'élaboration d'un cas dans le système CBR4WSD. Aussi, détaillons-nous notre modèle de description de service Web sémantiques couvrant aussi bien l'aspect fonctionnel que non-fonctionnel. Ensuite, nous introduisons notre modèle de cas qui demeure aligné avec les standards W3C. Nous exposons également les tâches de création et de préparation des cas et leur application dans notre système.

Le **sixième chapitre** présente la phase de remémoration du système CBR4WSD qui consiste à examiner la base de cas pour sélectionner les cas de services les plus similaires au problème posé. Nous étudions d'abord les problèmes liés à l'organisation de la base de cas avant de présenter l'approche adoptée qui retient les communautés de services comme critère d'organisation. Nous détaillons ensuite les opérations de maintenance des communautés assurées par l'administrateur et les mécanismes de préparation de la remémoration déclenchés au niveau du processus offline de la découverte. Les règles de vérification de la découvrabilité de services répondant à une requête, aussi bien que celles de découverte des cas sources candidats et de calcul des similarités fondé sur un appariement sémantique, sont ensuite décrites. Ce sont les fondements de l'étape de remémoration réalisée au niveau du processus online.

Le septième chapitre est consacré à la mise en œuvre et à la réalisation du système CBR4WSD. Il présente principalement les algorithmes de vérification de découvrabilité et de sélection des cas sources candidats, et ensuite ceux de découverte consistant en des fonctions de calcul de similarité relatives aux différentes parties descriptives d'un cas.

Enfin, la troisième partie composée du **chapitre 8** est consacrée à l'expérimentation et à l'évaluation de notre système. Nous y présentons les résultats d'une expérimentation visant à valider l'approche adoptée et à tester les algorithmes conçus par un jeu de test. Nous y présentons aussi les évaluations de performance de notre processus de découverte.

Enfin, nous concluons ce manuscrit par une synthèse des contributions et dégageons les perspectives de recherches.

Première partie

Etat de l'art

Chapitre 1

Le Raisonnement à Partir des Cas

SOMMAIRE

1.1. Introduction

Le Raisonnement à partir de cas (Case Based Reasoning, CBR) est un paradigme de raisonnement permettant de combiner la résolution de problèmes et l'auto-apprentissage. L'activité de recherche qui l'a accompagné depuis sa naissance lui a permis d'atteindre un niveau de maturité. Ainsi, s'est-il imposé parmi les techniques fondamentales de l'Intelligence Artificielle. Dans ce chapitre, nous exposons en détail le paradigme du Raisonnement à Partir des Cas (RàPC). Après avoir défini le RàPC, nous présentons ses principes. Le RàPC s'appuie sur un mode de raisonnement par analogie(s), ce qui nous amènera à présenter le carré d'analogie dans ce même contexte. Nous exposons également le cycle de raisonnement du RàPC et les phases qui en découlent. Nous présenterons aussi quelques techniques de recherche de similarité appliquées dans les systèmes RàPC.

1.2. Paradigme RàPC

1.2.1. Origine et définition

L'origine du RàPC vient des travaux sur la mémoire dynamique et l'apprentissage, menés par Schank à l'Université de Yale [Schank, 1982]. Ces travaux effectués dans le but de comprendre comment les individus mémorisent des informations et comment ils s'en rappellent, ont permis de conclure que les individus résolvent généralement les problèmes en se rappelant de la manière dont ils ont résolu des problèmes similaires dans le passé [Watson, 1999]. L'importance des expériences passées au cours d'une résolution a donné lieu au RàPC.

Selon Reisbeck et Schank [Reisbeck et Schank, 1989], le RàPC résout des problèmes en utilisant ou en adaptant des solutions de problèmes antérieurs. Beaubouchер précise qu'il consiste à retrouver des connaissances à partir d'épisodes passés dans un domaine particulier, connaissances qui partagent des aspects significatifs avec des expériences passées correspondantes, et à utiliser les connaissances transférées pour construire des solutions aux nouveaux problèmes ou pour justifier des solutions du domaine [Beaubouchер, 1994]. Watson quant à lui, définit le RàPC comme une méthodologie de résolution des problèmes qui utilise différentes technologies pour ce faire [Watson, 1999]. Il considère que le RàPC n'est pas une technologie en soi. Notamment, de nombreuses technologies ou approches peuvent être utilisées dans le cycle de RàPC que nous présentons plus loin.

En résumé, le RàPC est une méthodologie issue de l'intelligence artificielle et de la psychologie cognitive et orientée vers la résolution de problèmes. C'est un raisonnement par analogie qui consiste à résoudre un nouveau problème à partir d'expériences antérieures (problèmes connus et résolus précédemment), dont chacune constitue un cas. Le but est d'utiliser les connaissances issues des cas précédents et de les adapter pour résoudre de nouveaux problèmes. Chaque cas désigne un ensemble de problèmes associés à leurs solutions. Un «*cas source*» correspond à une expérience antérieure et peut servir en tant que cas similaire pour résoudre par analogie un problème en cours appelé «*cas cible*».

Le RàPC a émergé dans les années quatre-vingts. De nombreux auteurs ont contribué à son développement et l'ont appliqué dans des domaines variés tels que le domaine juridique, le génie civil, la chimie organique ou le diagnostic industriel. Aussi, de nombreux systèmes fondés sur le RàPC ont-ils vu le jour.

Parmi les systèmes précurseurs dans les années quatre-vingt, nous citons CYRUS [Kolodner, 1983], MEDIATOR [Simpson, 1985], CHEF [Hammond, 1986], PERSUADER [Sycara, 1987], CASEY [Koton, 1989].

La deuxième vague des systèmes développés dans les années quatre-vingt dix maintient l'orientation généraliste mais s'est principalement intéressée au domaine de la supervision et du diagnostic industriel. Les systèmes suivants sont représentatifs de cette période : JULIA [Hinrichs, 1992], le système juridique de GREBE [Branting, 1991], [Richter et Weiss, 1991], PAKAR [Watson et Abdullah, 1994], PAD'IM [Mille, 1995] [Fuchs et al., 1995], ROCADE [Fuchs, 1997], Resyn/RàPC [Lieber, 1997] [Lieber et Napoli, 1996] et le Déjà Vu [Smyth, 1996] [Smyth et Keane, 1993], [Smyth et Keane, 1995].

Les travaux des années deux mille ont ciblé des applications importantes telles que le décisionnel en cancérologie et les analyse d'accidents dans le domaine d'assurance. Entre autres, nous citons KASIMIR [Badra et al., 2006] dédié à la cancérologie et S3A [Ceausu et Despres, 2006] dans le domaine des assurances.

Outre ces systèmes, des travaux particuliers ont été présentés pour répondre à des besoins remarquables dans des domaines spécifiques tels que la découverte des services Web [De

Franco Rosa et De Oliveira, 2008] [Henni et al., 2013] [Osman et al., 2006 a] [Lajmi et al., 2011] [Sun et al., 2009] [Wang et Cao, 2007]. Ces approches feront l'objet d'une étude présentée dans le chapitre 3.

En outre, des plateformes spécialement dédiées à la conception de systèmes RàPC ont été proposées. A titre d'exemple, nous citons les plateformes jCOLIBRI [Bello-Tomas et al., 2004][Díaz-Agudo et al., 2007] et myCBR [myCBR, 2012] .

1.2.2. Carré d'analogie

Selon le carré d'analogie illustré dans la figure 1.1, le raisonnement par analogie adopté dans le RàPC vise à caractériser une situation en cours, appelée *cible*, en la mettant en correspondance avec une situation déjà rencontrée, appelée *source* [Mille et al., 1996]. Dans le cas où deux situations se ressemblent, ou sont analogues, des relations entre le problème cible et le problème source peuvent être établies et permettent par la suite de résoudre de façon similaire le problème cible.

L'idée de ce raisonnement (cf. figure 1.1) est de sélectionner d'abord, parmi les cas disponibles, un cas source similaire au problème cible en se référant aux valeurs de ses descripteurs (connaissances ou attributs décrivant un problème) et en appliquant une mesure de similarité Alpha. Les descripteurs constituant les éléments de la solution peuvent ensuite être adaptés, vu que les descripteurs décrivant le problème cible peuvent être similaires et non pas tout à fait identiques à ceux décrivant le problème source. Cette adaptation est effectuée conformément à des relations de dépendances Beta entre les valeurs des descripteurs du problème et celles des descripteurs de la solution. Autrement dit, les dépendances entre les descripteurs de la solution et ceux du problème dans un cas source sont par analogie projetées sur les descripteurs du cas cible.

En fonction de ces dépendances et des écarts alpha constatés entre les problèmes cible et source, l'adaptation permet de proposer une solution cible candidate qui pourra être évaluée par la vérification de sa conformité aux dépendances particulières qui pourraient exister entre le problème et la solution cibles (cf. figure 1.1) [Mille et al., 1996].

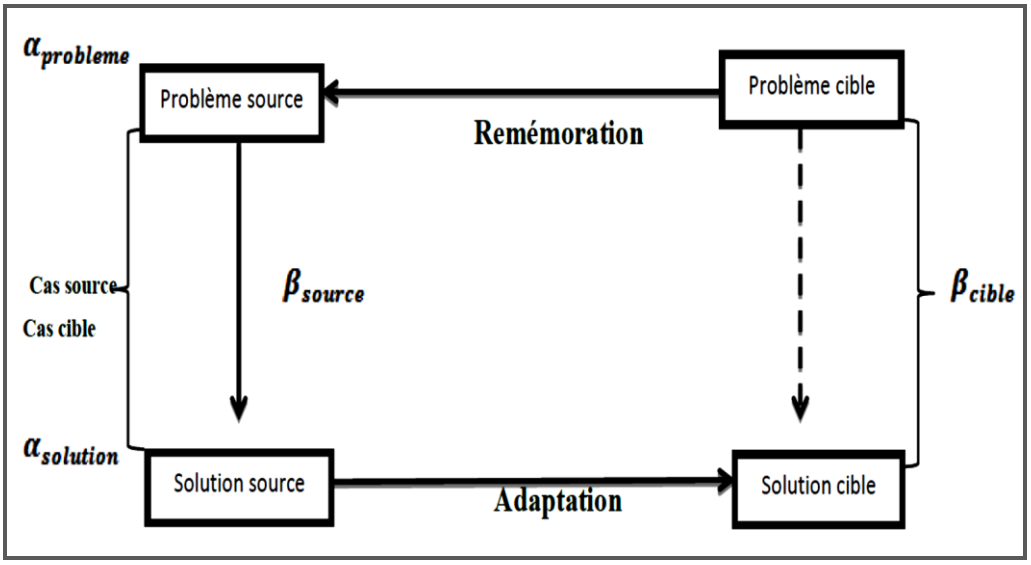


Figure 1.1 : Carré d'analogie [Source : Mille et al., 1996].

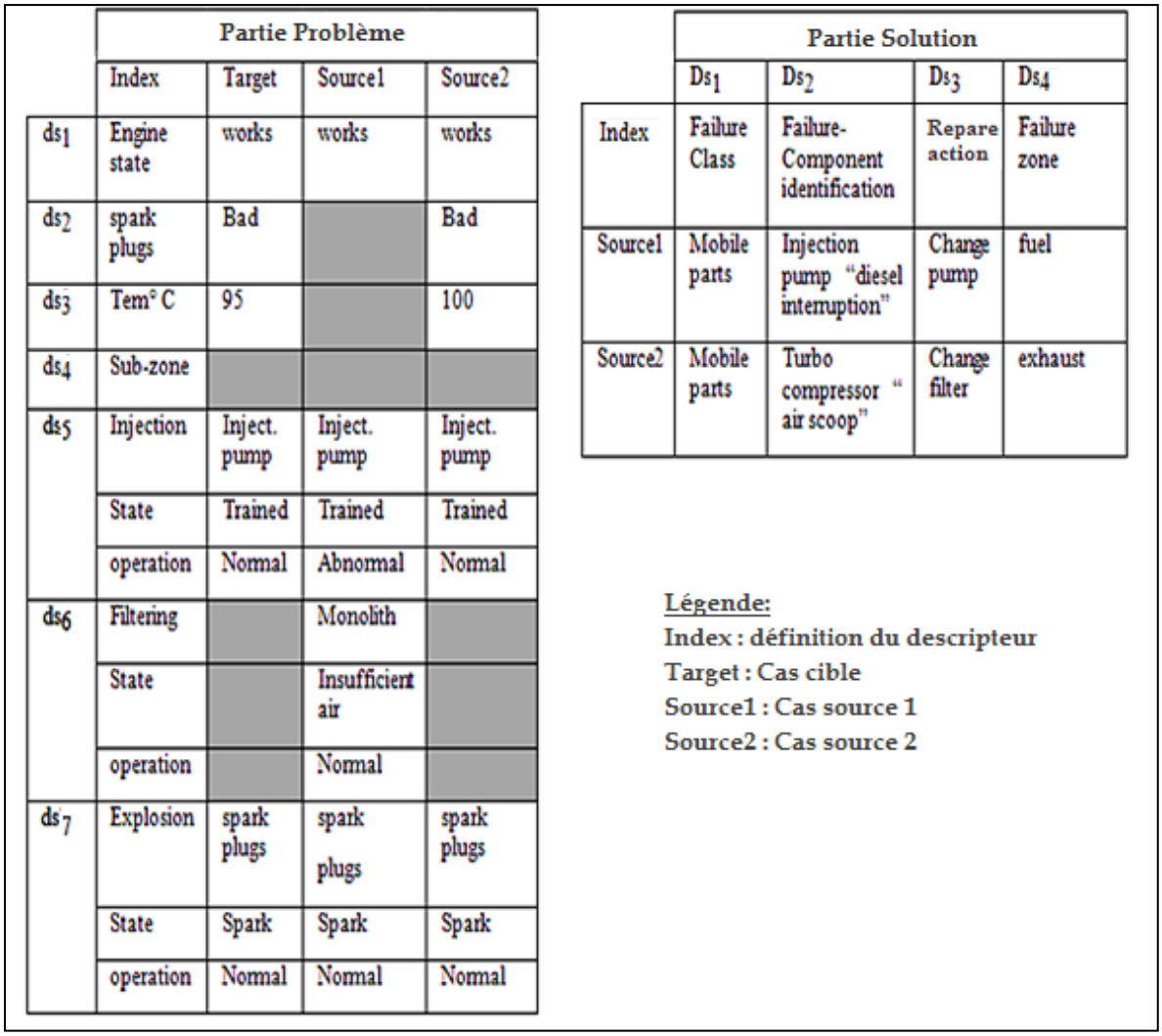
1.2.3. Représentation d'un cas

Un cas est décrit par deux parties : un problème noté *pb*, et sa solution notée *sol(pb)*. Chacune des deux parties est décrite par un ensemble de descripteurs. Un descripteur « d » est une connaissance représentée par une paire (a,v), où « a » est un attribut défini par un nom et « v » est la valeur qui lui est associée [Gebhardt et al., 1997]. L'attribut représente une caractéristique du domaine applicatif qui peut être numérique ou symbolique. Aussi, un cas constitue-t-il un tuple de descripteurs servant à décrire un problème et sa solution. Dans la suite de ce rapport, nous adoptons les conventions suivantes :

- un cas source est représenté par le tuple (d_1^S, …,d_n^S, D_1^S , …, D_m^S) où :
  - d_i^S est un descripteur du problème source.
  - D_j^S est un descripteur de la solution source.

- un cas cible est représenté par le tuple (d_1^C…, d_n^C , D_1^C, …, D_m^C) où :
  - d_i^C est un descripteur du problème cible.
  - D_j^C est un descripteur de la solution cible.

La figure 1.2 illustre un ensemble de cas manipulés dans un système RàPC pour le diagnostic industriel [El Bitar, 2010] [El Bitar et al., 2011b]. Chaque cas représente un diagnostic mesurant l'état de fonctionnement d'un moteur diesel effectué par un mécanicien superviseur. Celui-ci identifie les caractéristiques principales de la panne et les exprime sous forme de descripteurs d'un cas cible. Le système recherche alors les cas sources similaires pour lui proposer des solutions.



	Partie Problème			
	Index	Target	Source1	Source2
ds_1	Engine state	works	works	works
ds_2	spark plugs	Bad		Bad
ds_3	Tem° C	95		100
ds_4	Sub-zone			
ds_5	Injection	Inject. pump	Inject. pump	Inject. pump
	State	Trained	Trained	Trained
	operation	Normal	Abnormal	Normal
ds_6	Filtering		Monolith	
	State		Insufficient air	
	operation		Normal	
ds_7	Explosion	spark plugs	spark plugs	spark plugs
	State	Spark	Spark	Spark
	operation	Normal	Normal	Normal

	Partie Solution			
	Ds_1	Ds_2	Ds_3	Ds_4
Index	Failure Class	Failure-Component identification	Repare action	Failure zone
Source1	Mobile parts	Injection pump "diesel interruption"	Change pump	fuel
Source2	Mobile parts	Turbo compressor " air scoop"	Change filter	exhaust

Légende:
Index : définition du descripteur
Target : Cas cible
Source1 : Cas source 1
Source2 : Cas source 2

Figure 1.2 : Aperçu de cas manipulés dans un système RàPC pour le diagnostic industriel

Le premier tableau (cf. figure 1.2) présente les descripteurs (ds_1,…,ds_7) d'un cas cible (Target) et de deux cas sources (Source1 et Source2). A titre d'exemple, le descripteur ds_6 décrit la technique de filtrage (Monolith), l'état actuel reflétant le problème (Insuffisance d'air) et l'état de fonctionnement (Normal, Hors service, etc.). La description de chaque cas source est complétée dans le tableau 2 par la partie solution (cf. figure 1.2). Cette solution est également décrite en termes de descripteurs (Ds_1, Ds_2, Ds_3et Ds_4).

1.3. Cycle du RàPC

Le cycle classique de résolution de problèmes du RàPC a été proposé par Aamodt et Plaza [Aamodt et Plaza, 1994]. Ces auteurs ont proposé les quatre phases suivantes : Remémorer ou Retrouver, Réutiliser ou Adapter, Réviser et Mémoriser. Ce cycle a été amélioré et complété plus tard par Mille [Mille, 2006] (cf. figure 1.3), en ajoutant une phase supplémentaire avant

la remémoration, appelée « Elaboration ». La figure 1.3 résume l'enchaînement des cinq phases du cycle RàPC.

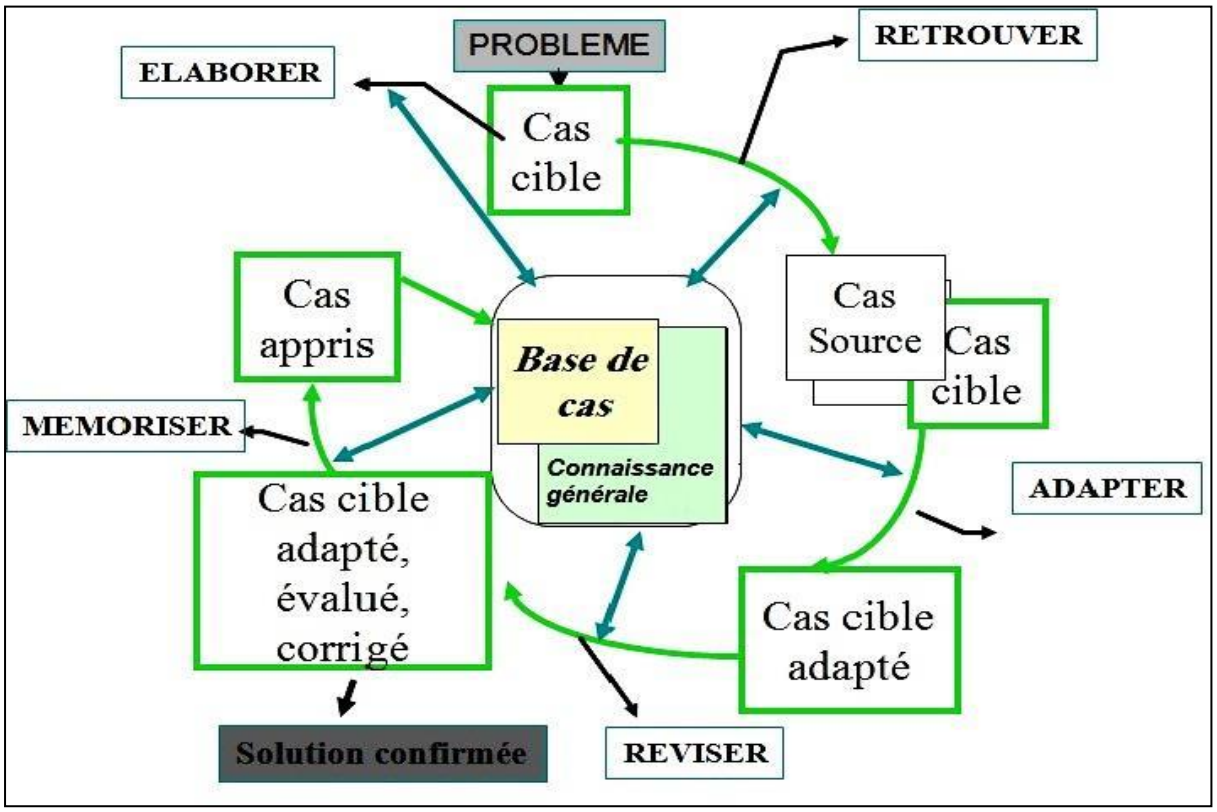


Figure 1.3 : Cycle du raisonnement à partir de cas [Source : Mille, 2006]

1.3.1. La phase d'élaboration

L'élaboration consiste en la formalisation de la description de problème à résoudre et sa mise en forme en vue de la remémoration.

Cette phase en amont consiste à construire d'abord le problème cible à partir de l'entrée du système du RàPC, tout en complétant les parties de sa description déductibles à partir des connaissances du domaine. Elle prépare ensuite le cas en identifiant les descripteurs influents dont la prise en considération fera que la recherche des cas proches puis leur adaptation soit efficace [Fuchs, 1997], [Fuchs et al., 2006]. L'importance des descripteurs influents identifiés lors de l'élaboration, est traduite par des poids utilisés pour calculer les mesures de similarité lors de la phase de remémoration. Cela permet en fait de sélectionner efficacement le cas le plus proche d'un cas cible, pour ensuite adapter la solution source en vue de la résolution du problème cible.

1.3.2. La phase de remémoration

La phase de remémoration est très importante dans un système de RàPC. Son objectif est de prouver et de sélectionner des cas sources relatifs à des problèmes similaires au problème cible. Une hypothèse majeure dans le RàPC est de considérer que des expériences de résolution de problèmes peuvent guider de futurs raisonnements pour la résolution de problèmes similaires et permettre la constitution de la base de cas par apprentissage. La remémoration est alors le processus qui permet d'extraire de la base de cas, les anciens cas dont la partie problème est similaire au problème à résoudre, en partant du principe que les cas similaires sont ceux utiles à la résolution du problème cible.

Les tâches de remémoration consistent à « apparier », c'est à dire les « évaluer et calculer leur similarité avec le cas cible» pour en identifier les cas similaires, et enfin à « sélectionner » le cas source le plus similaire. Dans la phase de l'appariement, une mise en correspondance des descripteurs de problèmes deux à deux s'effectue et elle est suivie par une évaluation qui permet d'estimer la similarité et l'ordonnancement des cas en conséquence. Des mesures de similarités sont alors à définir et appliquer aux descripteurs de la partie problème d'un cas. Les cas extraits de la base sont appelés les *cas sources.* Ces mesures de similarité cherchent la ressemblance entre les descripteurs des cas sources et ceux des cas cibles.

Cependant, déterminer la similarité entre deux cas est loin d'être une opération triviale. Elle implique l'utilisation de mesures et de connaissances de similarités fortement liées au domaine d'application. Plusieurs approches ont été développées dans ce cadre. Il y a, en premier lieu, « la similarité de surface » qui porte sur la ressemblance entre les attributs (descripteurs) des problèmes comparés qui peut s'étendre, éventuellement, aux attributs dérivés ou virtuels (calculés à partir de plusieurs attributs). Ce calcul de similarité est, surtout, valable pour les cas ayant une représentation attribut-valeur. Pour les cas ayant une représentation plus complexe, « la similarité structurelle » portant sur la ressemblance des relations des problèmes comparés est la mieux indiquée.
On distingue deux types de similarité : « locale » et « globale ». La mesure de similarité locale détermine la similarité entre les valeurs de deux attributs simples ou complexes. Lorsque les attributs sont complexes, elle est calculée à partir des similarités locales des attributs simples qui les composent. Enfin, les valeurs des similarités locales sont agrégées pour obtenir la similarité globale de deux cas comparés.

Par ailleurs, les attributs d'un cas cible n'ont pas toujours la même importance dans le calcul de similarité. Ainsi, il est important de permettre à l'utilisateur d'associer à chaque attribut un certain poids dans l'intervalle [0, 1]. Nous expliquerons dans la section 4 des détails sur les techniques de recherche des cas similaires dans le cadre du RàPC.

1.3.3. La phase de réutilisation

La phase de réutilisation des cas est le processus proposant une solution à un nouveau problème à partir des solutions appartenant aux cas sources remémorés [Lopez de Mantaras et al., 2005].

Cette réutilisation consiste en une simple copie ou une adaptation. Dans le cas de copie, la solution ou la méthode de solution des cas remémorés est transférée au nouveau cas. Dans le cas d'adaptation, la solution ou sa méthode est modifiée lorsque la solution source n'est pas tout à fait pertinente. En effet, lorsque plusieurs cas sont remémorés, une solution pourra être dérivée de plusieurs cas ou bien plusieurs alternatives pourront être proposées. Il existe alors trois façons de réutiliser des cas remémorés [Armaghan, 2009]:

- La réinstantiation : Lorsque le degré de la similarité entre le cas cible et le cas source remémoré est élevé et qu'il n'y a pas de contrainte imposée sur la solution. La réinstantiation est la façon la plus simple en phase de réutilisation. La solution pour un nouveau problème est copiée directement, sans modification, à partir du cas remémoré.
- L'adaptation substitutionnelle : la substitution remplace la partie du descripteur de la solution remémorée qui ne répond pas aux besoins du nouveau problème.
- L'adaptation transformationnelle : la transformation est utilisée lorsqu'il n'y a aucune substitution appropriée possible. Une solution pertinente sera dérivée en prenant en considération les contraintes et les caractéristiques de la solution demandée. Par exemple : il est difficile parfois de trouver un remplaçant d'un élément qui répond aux exigences du problème. Dans cette situation, la solution source devrait être ajustée (partiellement ou entièrement).

Les étapes suivantes sont importantes dans l'adaptation transformationnelle :

- La remémoration des cas similaires à partir de la base de cas,
- La réparation des solutions sources en contrôlant la structure sémantique des substitutions disponibles.

- Si aucune substitution n'est disponible, il faut transformer la solution source en remplaçant certaines parties par des composants pertinents.
- Ajouter la nouvelle solution à la structure sémantique pour des utilisations ultérieures [Pal et al., 2004].

1.3.4. La phase de révision

La solution obtenue à la sortie de la phase de réutilisation doit être testée pour vérifier si elle convient comme solution pour le problème traité. La révision permet de prendre en compte les échecs non-prévus. Elle ouvre la voie à un processus d'apprentissage permettant de compléter ou de corriger les connaissances du système ayant mené à cet échec [Fuchs, 2008].
Globalement, deux parties sont nécessaires à la réalisation de cette phase : l'évaluation de la solution produite par adaptation, et éventuellement, la réparation de la solution en réutilisant les connaissances du domaine spécifique concerné.

Au cours de cette phase, la solution proposée à l'issue de la phase de réutilisation sera évaluée. Cette évaluation est généralement externe au système RàPC. Mille propose pour réaliser cette phase, de procéder par exemple comme suit [Mille, 2006] :

- Suivant le domaine, on peut faire appel à un simulateur ou à un système expert pour évaluer la solution cible.
- Ou tester la solution dans le « monde réel ». Toutefois, la durée de l'évaluation dans ce cas peut être très longue, notamment dans le domaine médical pour le test de traitements.

Si l'évaluation est satisfaisante, la nouvelle solution sera retenue pour la phase de mémorisation et apprentissage, sinon il faut la réparer. La réparation consiste à déterminer les raisons de l'échec et intégrer les modifications nécessaires.

1.3.5. La phase de mémorisation et d'apprentissage

Lorsqu'un utilisateur valide un cas, il est essentiel de le sauvegarder dans la base de cas pour une réutilisation ultérieure. Il s'agit du stockage du nouveau cas pour l'enrichissement de la base de cas. Toutefois, le nouveau cas peut être très similaire à un cas source existant, ainsi la mémorisation de ce cas peut conduire à une duplication des informations de la base de cas. Il faut donc prendre en compte quelques considérations avant de mémoriser un nouveau cas.

Selon Bénard et De Loor [Bénard et De Loor, 2008], il est nécessaire d'ajouter un cas lorsque :

- Le contexte de ce dernier ne peut pas être complètement identifié en se référant à celui d'un cas de la base.
- L'objectif du contexte du cas concerné est différent de celui du cas dans la base auquel il est identique.
- L'action pour atteindre l'objectif du cas concerné est nouvelle, elle n'est pas présente dans les actions permettant d'atteindre cet objectif dans la base de cas.

Toutefois, ces préconisations sur la résolution d'un problème dans un système du RàPC ne sont pas obligatoires. Généralement, si la solution a été générée à partir d'un ancien cas source, un nouveau cas est créé dans la base de cas ou bien un ancien cas se trouvera généralisé afin de subsumer le nouveau cas. Par contre, si le problème est résolu sans l'aide des cas préexistants, par exemple à l'aide des connaissances d'un expert, un cas complètement nouveau sera créé et stocké dans la base.

L'apprentissage par l'expérience est primordial pour un système RàPC. En effet, il permet au système de développer sa capacité de résolution au fur et à mesure de l'ajout de nouveaux cas dans la base de cas. Néanmoins, la maintenance de la base de cas doit être assurée. Cette maintenance consiste à développer des techniques pour contrôler les changements de cas et y réagir, conformément à différentes sources de connaissances [Armaghan, 2009]. Ces sources sont :

- *Le vocabulaire* qui correspond aux informations sur les définitions et les structures utilisées dans le système de RàPC.
- *Les mesures de similarité* qui correspondent aux mesures nécessaires pour la recherche des cas.
- *Les règles d'adaptation* qui correspondent aux règles de transformation de la solution.
- *La base des* cas qui rassemble et organise les cas [Haouchine et al., 2006].

Les mesures de similarité sont cruciales aussi bien dans le cycle RàPC en général que dans notre propre recherche. Nous les étudions dans la section suivante.

1.4. Similarité

Le principe du RàPC consiste à ne pas se limiter à rechercher dans la base de cas un cas répondant exactement à une requête mais au contraire d'identifier un ensemble de cas « proches » susceptibles d'apporter des éléments de réponses.

En effet, la fonction de similarité qui détermine la « proximité » de deux cas est la clé de voute du RàPC. Nous introduirons cette notion globalement dans le domaine informatique avant de l'étudier dans le contexte du RàPC à travers les notions de similarités locales et globales.

1.4.1. Qu'est-ce qu'une fonction de similarité ?

Dans tous les domaines de l'informatique dans lesquels on désire analyser de manière automatique un ensemble de données, il est nécessaire de disposer d'un opérateur capable d'évaluer précisément les ressemblances ou les dissemblances qui existent au sein de ces données. Sur cette base, il devient alors possible d'ordonner les éléments d'ensemble, de les hiérarchiser ou encore d'en extraire des invariants. Pour qualifier cet opérateur, nous utiliserons dans ce chapitre le terme de *fonction de similarité* ou plus simplement celui de *mesure de similarité*.

Exprimées sous des formes multiples, des mesures de similarité sont mises en œuvre dans de nombreux domaines. C'est notamment le cas de l'analyse des données, la reconnaissance des formes, l'apprentissage symbolique, ou encore les sciences cognitives. Même s'il peut sembler difficile au premier abord d'établir un lien entre ces différentes approches, on peut donner une définition commune de ce qu'est une similarité ainsi que les principales tâches qu'elle permet d'effectuer.

De manière générale, une fonction de similarité est définie dans un univers *U* qui peut être modélisé à l'aide d'un quadruplet (Ld, Ls, T, FS) [Lenz, 1999] où :

- Ld le langage de représentation utilisé pour décrire les données,
- Ls le langage de représentation de la similarité,
- T un ensemble de connaissances que l'on possède sur l'univers étudié,
- FS la fonction binaire de similarité, telle que FS : Ld×Ld×Ls.

D'après Lenz, lorsque la fonction de similarité a pour objet de quantifier les ressemblances entre les données, le langage Ls correspond à l'ensemble des valeurs dans l'intervalle [0..1], et c'est dans ce cas que l'on parle de mesure de similarité. Il considère schématiquement que ces mesures interviennent dans trois types de traitement de données, à savoir la classification, l'identification et la caractérisation [Lenz, 1999].

- Le processus de classification vise à structurer les données contenues dans *U*, en fonction de leurs ressemblances, sous la forme d'un ensemble de classes à la fois homogènes et contrastées;
- Le processus d'identification a pour but de déterminer la classe à laquelle un objet inconnu est susceptible d'appartenir, ou encore, de trouver à quel(s) objet(s) de *U* il est le plus ressemblant;
- Le processus de caractérisation permet de construire une représentation explicite des informations qui sont communes à un ensemble de données. Dans ce cas, le langage Ls est souvent un sous-ensemble de Ld (Ls ∈ Ld).

1.4.2. La similarité dans le RàPC

Rappelons qu'un système de RàPC doit résoudre, dans un domaine particulier, de nouveaux problèmes en adaptant des solutions préexistantes qui ont été déjà utilisées pour la résolution d'anciens problèmes.

La remémoration de cas sources (problèmes déjà résolus) pour chercher une solution à un nouveau cas cible (nouveau problème) est le point focal du cycle RàPC. En effet, les systèmes de RàPC utilisent différentes techniques pour comparer une description d'un cas cible avec l'un des cas sources déjà connus. L'utilisateur donne une description du nouveau problème et le système cherche dans sa base de cas le cas source dont la description est la plus similaire à la description du nouveau problème cible.

Ensuite, à travers une session de consultation des questions et réponses des cas sources, le système retourne les cas candidats, ce qui permet à l'utilisateur de sélectionner le cas qui semble être le plus approprié (cf. figure 1.4).

C'est enfin dans le contexte de la remémoration que des mesures de similarité sont utilisées pour identifier les cas sources similaires à un cas cible. Les degrés de similarité de ces cas doivent dépasser nécessairement un seuil défini par un expert du domaine concerné.

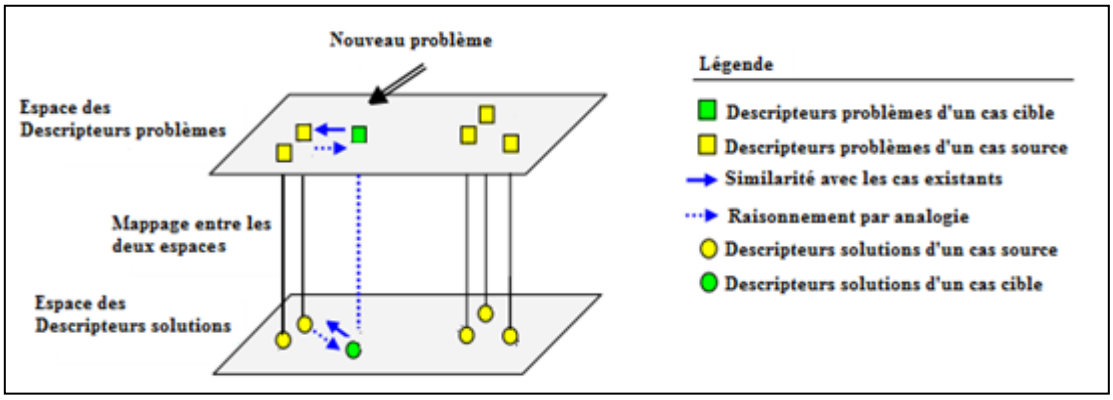


Figure 1.4 : Relation entre l'espace problème et l'espace solution dans le RàPC (adapté) [Leake, 1996]

Nous notons que les cas sont décrits par des paramètres quantitatifs et qualitatifs, appelés caractéristiques ou tout simplement attributs auxquels sont associés des valeurs (cf. figure 1.2). L'algorithme de raisonnement à partir de cas détermine la similarité entre les cas en se basant sur ces valeurs. Par ailleurs, une mesure de similarité dans le RàPC doit être :

- Réflexive car un cas est similaire à lui-même.
- Symétrique i.e. si A est similaire à B alors B est aussi similaire à A.
- Non-transitive. Une mesure de similarité n'est pas toujours transitive. Si A est similaire à B et B est similaire à C, alors A n'est pas forcement similaire à C, car les caractéristiques qui définissent les similarités entre A et B et entre B et C ne sont pas nécessairement les mêmes.

Les mesures de similarité cherchent des correspondances entre les descripteurs problèmes des cas sources et ceux du cas cible. L'objectif de ces mesures de similarité est de retrouver dans la base de cas, le cas similaire au problème actuel dans le sens qu'il soit facilement adaptable à ce nouveau problème. Ainsi, le degré de similarité est la fonction du rapport utilité/adaptabilité de la solution.

Par ailleurs, comme nous l'avons déjà mentionné, il existe deux types de mesures de similarité : locales ou globales. Nous en présentons les détails dans les sections 4.3 et 4.4.

1.4.3. Similarités locales

Les mesures de similarités locales concernent les caractéristiques du cas. Elles dépendent du type des caractéristiques et des rangs des valeurs des caractéristiques. La similarité locale peut être calculée pour des valeurs numériques, par exemple, comparer une lecture de température de 20°C et 22°C. Elle peut aussi être calculée entre deux valeurs non-numériques, par exemple le type de la boîte à vitesse de deux véhicules.

La similarité locale dépend de l'intervalle de valeurs acceptées, par exemple, dans l'intervalle [-100, +100], les nombres 30 et 40 ne sont pas similaires, alors que dans l'intervalle [-10000, +10000], ces deux valeurs sont considérées comme similaires. Plus généralement, le calcul des similarités locales dépend du type de descripteur et il est basé sur la distance [Haouchine, 2010]. Nous notons à titre d'exemples que :

- pour les valeurs de descripteurs numériques:

  $\text{sim}(a,b) = 1 - \frac{|a-b|}{range}$

- pour les valeurs de descripteurs symboliques (mono-valeurs) :

  $$\text{sim}(a,b) = \begin{cases} 1 \text{ pour } a=b \\ 0 \text{ pour } a \neq b \end{cases}$$

- pour les valeurs de descripteurs symboliques (multi-valeurs) :

  $\text{sim}(a, b) = \frac{card(a) \cap card(b)}{card(a \cup b)}$

  Où :

  a et b : sont des valeurs des descripteurs.

  card (x): est la cardinalité de l'ensemble x.

  range : est la valeur absolue de la différence entre la borne supérieure et la borne inférieure de l'ensemble des valeurs considéré.

Ces mesures de similarité sont appliquées dans différents domaines du RàPC, plus spécifiquement dans les domaines du diagnostic industriel et médical. La comparaison des valeurs n'est pas toujours directement réalisable surtout lorsque le référentiel lexical n'est pas commun. Par exemple, pour comparer les valeurs « high » et « élevé », on a besoin d'une mise en correspondance sémantique entre ces valeurs. Notamment, dans le cas de services

Web sémantiques dont la description met en jeu des concepts d'ontologie, c'est plutôt la distance sémantique entre ces concepts dans l'ontologie qui est adoptée pour la mesure de similarité, et que nous adoptons également dans ce travail. Ainsi, nous en présentons les détails dans le chapitre 6.

1.4.4. Similarité globale

La similarité globale est calculée au niveau des cas ou des objets en agrégeant les similarités locales. Différentes mesures de similarité globale sont utilisées dans les systèmes de RàPC. Leur utilisation dépend du domaine d'application. Calculer la similarité globale entre deux cas se ramène généralement à calculer la moyenne des similarités locales (calculer la somme et la diviser par le nombre d'attributs communs). Pour prendre en considération l'influence de certains attributs par rapport à d'autres, il est possible de les pondérer en associant un coefficient numérique à chaque attribut.

Nous donnons ci-après quatre mesures fréquentes de similarité globale entre deux cas A et B, à n attributs chacun.

Soient :

- n le nombre d'attributs,
- w_i le poids du $i^{\text{ème}}$ attribut (évalué en fonction de l'importance), avec $\sum_{i=1}^{n} w_i = 1$
- $sim_i(a_i, b_i)$ est la similarité locale calculée pour l'attribut i et appliquée aux valeurs a_i. et b_i de cet attribut.

- *Block city*

$$Sim\ (A,B) = \sum_{i=1}^{n} sim_i\ (a_i, b_i)$$

- *Weighted Block city*

$$Sim\ (A,B) = \sum_{i=1}^{n} w_i sim_i\ (a_i, b_i)$$

- *Minkowski*.

$$Sim\ (A,B) = [\frac{1}{n}\sum_{i=1}^{n}[sim_i(a_i, b_i)]^r]^{\frac{1}{r}}$$

Cette mesure est dite :

Distance *Euclidienne* lorsque r = 2

Distance de *Manhattan* lorsque r = 1

- *Weighted Minkowski*.

$$Sim\ (A,B) = [\frac{1}{n}\sum_{i=1}^{n} w_i[sim_i(a_i, b_i)]^r]^{\frac{1}{r}}$$

Cette mesure est dite :

Distance *Weighted Euclidienne* lorsque r = 2

Distance de *Weighted Manhattan* lorsque r = 1

Dans le cas où $\sum_{i=1}^{n} w_i \neq 1$, il faut tout simplement diviser les formules pondérées par la somme des poids des attributs [Haj Said, 2004]. Ainsi, l'expression de la distance *Weighted Block-City* devient :

$$Sim\ (A,B) = \frac{\sum_{i=1}^{n} w_i sim_i(a_i,b_i)}{\sum_{i=1}^{n} w_i}$$

Nous notons que la façon dont les attributs sont pondérés est cruciale pour que la mesure effectuée soit pertinente. Aussi, cette pondération est-elle souvent confiée à un expert du domaine.

1.5. Synthèse

Dans ce chapitre, nous avons présenté le paradigme de Raisonnement à Partir des Cas à travers son origine, ses principes et son cycle en explicitant chacune de ses phases.

En effet, la remémoration de problèmes passés et résolus, dits aussi cas sources, pour rechercher une solution à un nouveau problème ou cas cible, est une phase cruciale dans le cycle RàPC.

La remémoration s'appuie sur des mesures de similarité pour identifier les cas sources similaires relatifs à un cas cible. Différentes techniques d'évaluation de la similarité dans un système RàPC ont été proposées. Elles adoptent différentes mesures telles que : Block city, Euclidienne, Manhattan, etc.

Les mesures les plus connues et les plus utilisées demeurent sans contestation la distance euclidienne et la distance de Manhattan, qui ne sont en fait que des cas particuliers de la mesure de Minkowski.

Chapitre 2

Les approches de description des services Web

SOMMAIRE

2.1. Introduction

Avec l'augmentation des défis de l'entreprise qui doit aujourd'hui pallier aussi les problèmes majeurs d'adaptation de ses systèmes dans des environnements flexibles et évolutifs, l'architecture SOA (*Service-Oriented Architecture*) plus spécifiquement les services Web qui sont une implémentation particulière de cette architecture, ont été procurés comme solution à ces problèmes. Les services Web sont des composants logiciels interopérables permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués.

Le W3C (World Wide Web Consortium) qui est l'organisme de standardisation d'Internet, propose de décrire syntaxiquement les services Web en utilisant le standard de description WSDL (Web Service Description Language) et de les publier dans l'annuaire UDDI (Universal Definition and Description Integration). Cet annuaire met à la disposition de l'utilisateur un ensemble d'informations décrivant les propriétés du service recherché. Notamment, il indique ce que fait le service et la façon d'y accéder. Cependant, en plus de ces propriétés fonctionnelles, un service Web peut posséder des propriétés non-fonctionnelles qui font de ce composant logiciel, le service le plus adapté à un client donné plutôt qu'à un autre.

Toutefois, le mécanisme de découverte soutenu par le standard UDDI qui fournit une interface de recherche syntaxique basée sur les mots-clés, n'est pas assez puissant pour la découverte automatisée des services. Ceci est dû au manque de considération de la sémantique dans le processus de découverte adopté. En effet, d'éventuels problèmes d'hétérogénéités sémantiques peuvent apparaitre aux moments de la découverte, de la composition et même de l'invocation. La réponse à de telles exigences d'interopérabilité sémantique a été apportée par les services Web sémantiques qui combinent le web sémantique et la technologie des services Web afin de permettre une interaction automatique et dynamique entre les systèmes. Ainsi, les machines peuvent coopérer grâce à des interfaces de services Web décrites explicitement avec une sémantique basée sur des ontologies de référence. Ceci offre aux services Web, d'une part, une description sémantique 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 la découverte, de l'invocation et de la composition [Berners-Lee et al., 2001].

Des approches sémantiques ont été alors proposées. Ces approches couvrent bien la dimension sémantique mais elles ne mettent pas l'accent sur les propriétés non-fonctionnelles. Elles considèrent notamment un service Web sémantique comme un service Web dont la description intègre, en plus de la description fonctionnelle, une dimension sémantique. Cette dimension englobe toute description complémentaire de la description fonctionnelle (comme la catégorie de service et les objectifs du service Web, etc.). Elle permet à des agents logiciels de lire la description d'un service Web pour déterminer si celui-ci fournit les fonctionnalités désirées.

Dans ce contexte, nous considérons que l'efficacité du processus de découverte dépend particulièrement du degré d'expressivité des descriptions des services Web. Aussi, ce chapitre est-il consacré à l'état de l'art de la description des services Web. Nous exposons d'abord les approches proposées dans la littérature pour la description des services Web. Ces approches décrivent les services Web à deux niveaux syntaxique et/ou sémantique et peuvent être essentiellement réparties d'une part, en standards W3C, à savoir WSDL 2.0 [Chinnici et al., 2007], SAWSDL (Semantic Annotation for WSDL) [Farell et Lausen, 2007a] et WS-Policy 1.5 (Web Service Policy) [Vedamuthu et al., 2007], et d'autre part en modèles sémantiques tels que OWL-S (Ontology Web Language for Service) [Martin et al., 2004] et WSMO (Web Service Modeling Ontology) [Roman et al., 2006]. Nous présentons ensuite une analyse comparative de ces approches qui a fait l'objet d'un article de synthèse [El Bitar et al., 2013a].

Nous clôturons ce chapitre par une synthèse qui souligne l'intérêt particulier des standards W3C et justifie leur choix par la suite comme fondement de notre approche de découverte de services Web.

2.2. Approches de description de services Web

Nous distinguons deux niveaux de description pour les services Web à savoir syntaxique et/ou sémantique. Les spécifications introduites par [Chinnici et al.,2002] ont mis l'accent sur le besoin fonctionnel décrit à un niveau purement syntaxique.

Le langage WSDL est notamment un standard W3C utilisé comme langage de base dans l'industrie pour la description des services Web implémentant des architectures du SOA. Il est limité au niveau de description syntaxique. Pour pallier ce problème, les standards SAWSDL et WS-Policy du W3C ont été proposés. Ceux-ci enrichissent les fichiers WSDL non

seulement en considérant le niveau sémantique dans la description des services mais aussi en tenant compte de l'aspect non-fonctionnel.
Dans le même contexte, des modèles sémantiques [Fensel et al., 2002] [Martin et al., 2004][Akkiraju et al., 2005][Battle et al., 2006][Badr et al., 2008] ont été proposés pour couvrir la dimension sémantique dans la description des services Web.

L'analyse des caractéristiques de chacune des approches qui demeurent les plus utilisées dans ce cadre, à savoir, les standards WSDL 2.0, SAWSDL et WS-Policy 1.5, ainsi que OWL-S et WSMO, a permis de distinguer deux familles [Omrana, 2014] :

- **Les approches à base de langage syntaxique.** Ces approches décrivent syntaxiquement les services Web. Elles regroupent le standard WSDL et ses extensions SAWSDL et WS*- Spécifications tels que WS-Policy. En plus de leur qualification de standard W3C, ces langages utilisent des structures de format XML, ce qui permet de favoriser leur extensibilité.
- **Les approches à base de modèles sémantiques.** Ces approches sont fondées sur des langages particuliers d'ontologie du Web. Ceci leur permet de décrire les services Web de façon non ambiguë et interprétable par des programmes. Cette famille comprend OWL-S et WSMO qui représente les modèles sémantiques les plus utilisés dans la littérature.

Nous présentons les détails de chacune de ces approches dans les sections suivantes.

2.3. Les standards W3C: WSDL, SAWSDL et WS-Policy

Le W3C est un organisme international de normalisation qui, entre autres, œuvre pour promouvoir et superviser le développement des standards de services Web. Dans ce contexte, ce consortium recommande les standards WSDL 2.0, SAWSDL et WS-Policy 1.5 pour la description des services Web. Nous présentons les notions relatives à ces standards dans la suite de cette section.

2.3.1. WSDL

WSDL (Web Service Description Language) est un langage de description des services Web au format XML. Il repose essentiellement sur les protocoles standards de communication SOAP (Simple Object Access Protocol) et HTTP (HyperText Transfer Protocol) pour

l'invocation d'objets distants. WSDL utilise une grammaire XML bien structurée pour décrire les services Web et paramétrer les échanges de messages [Bray et al., 2006].
WSDL 1.1 a été proposé en 2001 au W3C pour standardisation. La version 2.0 a été recommandée en 2007 et elle est désormais une recommandation officielle du W3C. Le standard WSDL est caractérisé par la possibilité d'extension et d'ajout d'éléments et/ou d'attributs (pour intégrer par exemple l'aspect sémantique dans la description du service). Cette extension est spécifiquement exploitée par SAWSDL (voir la section 2.3.2).

Le standard WSDL favorise les communications inter-applications grâce à son approche de description qui inclut la spécification des éléments comme le protocole de communication (par exemple SOAP) et les types des données échangées, ce qui est nécessaire pour l'interaction avec le service Web. Le W3C a défini notamment les catégories d'informations à prendre en compte dans la description d'un service Web. Les éléments décrits dans WSDL sont principalement les suivants :

- les opérations proposées par le service Web;
- les données et messages échangés lors de l'appel d'une opération ;
- le protocole de communication ;
- les ports d'accès au service.

Le standard WSDL offre une description sur deux niveaux : une description abstraite décrivant la fonction abstraite offerte par un service et une description concrète décrivant comment et où accéder à cette fonction [Chinnici et al., 2007]. Le niveau abstrait est utilisé principalement lors du processus de sélection tandis que le niveau concret est plutôt utilisé lors de l'invocation des opérations du service Web.

Description abstraite :
Le niveau abstrait décrit les informations propres aux opérations proposées par le service ainsi que les informations traitant les messages et les données échangés lors de l'invocation du service. Ce niveau décrit les informations suivantes :

- **Les types de données** : Le document WSDL permet de décrire les types de données échangées par un schéma XML (au moyen de la balise "`types`"). WSDL supporte les types élémentaires (tels que les entiers et les chaînes de caractères) et les types complexes. Si les données échangées possèdent une structure particulière (*i.e.* un type

complexe), il est possible de les décrire par l'intermédiaire d'un schéma XML [Thompson et al., 2004].

- **Les opérations** : Une opération représente une fonctionnalité (une méthode) proposée par le service Web décrit. Chaque opération notée "`operation`" est identifiée par son nom et les messages d'entrée (`Input message`), de sortie (`Output message`) ou d'erreur (`fault message`) qu'elle peut échanger.
- **Les messages** : Un message correspond aux données qui sont véhiculées selon les opérations invoquées. Chaque opération du service possède au plus deux messages d'entrée et de sortie : le premier correspond à la requête tandis que le second correspond à la réponse. Elle peut posséder également un message d'erreur.

Un client souhaitant invoquer un service Web particulier doit en plus des informations contenues dans le niveau abstrait, connaître comment et où les messages échangés seront envoyés. Cet échange de message se fait conformément aux spécifications de la partie concrète du modèle WSDL notées à travers des éléments "`binding`".

Description concrète :

Les principales informations décrites au niveau concret sont les suivantes :

- **Le protocole de communication** : La description des protocoles de communication permet de définir le protocole à utiliser pour l'appel des méthodes du service. Si nécessaire, le document WSDL peut contenir autant de descriptions de protocoles que d'opérations, étant donné que le protocole de communication peut différer selon chaque opération du service.
- **Les ports d'accès au service** : Dans un document WSDL, l'accès au service est défini par une collection de ports d'accès appelés aussi "`endpoints`". Chaque port représente la localisation du service (*i.e.* son URL). En effet, un élément XML noté "`service`" regroupe les ports d'accès correspondant à une même interface [Chinnici et al., 2007]. Ainsi, un même service Web peut être accessible depuis plusieurs ports.

Les informations contenues dans WSDL correspondent essentiellement à la description du profil fonctionnel du service. Avec WSDL, le client peut invoquer le service en se référant aux informations dans son fichier WSDL, renseignant sur sa description abstraite (méthodes

disponibles, paramètres d'entrée et sortie, etc.) et sa description concrète (description des protocoles de communication, des points d'accès au service, etc.).

Dans la suite, nous illustrons l'utilisation de la version WSDL 2.0 par un exemple tiré de la description d'un service Web nommé *GlobalWeather* proposé par le fournisseur *webserviceX.net*. Ce service renvoie un ensemble d'informations concernant la météorologie (vitesse du vent, visibilité, condition météorologique, température, etc.) selon la ville et le pays donnés en paramètres d'entrée.

Exemple illustratif en WSDL 2.0

La structure générale de WSDL 2.0 est composée d'un élément racine (`description`) et de quatre sous-éléments principaux (`types`, `interface`, `binding` et `service`) comme l'illustre la figure 2.1.

```
<description targetNamespace="xs:anyURI">
        [ <import namespace="xs:anyURI" location="xs:anyURI" ? />
        | <include location="xs:anyURI" /> ] *
<types>
    [ <xs:import namespace="xs:anyURI" schemaLocation="xs:anyURI" ? /> |
      <xs:schema targetNamespace="xs:anyURI" ? /> |
      other extension elements ] *
</types>?
<interface>
    <operation name="xs:NCName" pattern="xs:anyURI"? style="list of xs:anyURI" ? >
        <input messageLabel="xs:NCName"? element="union of xs:QName, xs:token"? />
        <output messageLabel="xs:NCName"? element="union of xs:QName, xs:token"? />
        <infault ref="xs:QName" messageLabel="xs:NCName"? />*
        <outfault ref="xs:QName" messageLabel="xs:NCName"? />*
    </operation>
</interface>
<binding name="xs:NCName" interface="xs:QName"? type="xs:anyURI" >
    <fault ref="xs:QName" />
    <operation ref="xs:QName" >
        <input messageLabel="xs:NCName"? />
        <output messageLabel="xs:NCName"? />
        <infault ref="xs:QName" messageLabel="xs:NCName"? />
        <outfault ref="xs:QName" messageLabel="xs:NCName"? />
    </operation>
</binding>
<service name="xs:NCName" interface="xs:QName" >
    <endpoint name="xs:NCName" binding="xs:QName" address="xs:anyURI"? />
</service>
</description>
```

Figure 2.1 : Eléments composant un document WSDL 2.0.

L'élément « `description` » est l'élément racine d'un document WSDL 2.0. Il est utilisé pour déclarer les espaces de nom utilisés tout au long du document (cf. figure 2.2). Dans l'exemple illustré, l'espace de nom xmlns= « http://www.w3.org/ns/wsdl » est l'espace de nom utilisé par défaut pour désigner l'espace des éléments WSDL.

```
<description
    xmlns:s= "http://w3.org/ns/wsdl"
    xmlns:tns= "http://www.webserviceX.net"
    targetNamespace= "http://www.webserviceX.NET">
```

Figure 2.2: Structure de l'élément *description* dans la description du service GlobalWeather en WSDL 2.0.

L'élément « `types` » décrit les types de messages échangés par le service lors de l'appel de l'une des méthodes. Dans notre exemple (cf. figure 2.3), nous remarquons que les messages échangent quatre types structurés (GetWeather, GetWeatherResponse, GetCitiesByCountry et GetCitiesByCountryResponse) et un type simple nommé string de type string. A titre d'exemple, le type complexe GetWeather est composé de deux éléments (nommés respectivement CityName et CountryName) qui sont de type string.

```
<types>
    <s:schema xmlns:http= "http://schemas.xmlsoap.org/wsdl/http/"
              xmlns:soap= "http://schemas.xmlsoap.org/wsdl/soap/"
              xmlns:s= "http://w3.org/2001/XMLSchema/"
              xmlns:soapenc= "http://schemas.xmlsoap.org/soap/encoding/"
              xmlns:tm= "http://microsoft.com/wsdl/mime/textMatching/"
              xmlns:mime= "http://schemas.xmlsoap.org/wsdl/mime/"
              xmlns:wsdl= "http://schemas.xmlsoap.org/wsdl/"
              elementFormDefault="qualified"
              targetNamespace= "http://www.webserviceX.NET">
        <s:element name="GetWeather">
            <s:complexType>
                <s:sequence>
                    <s:element minOccurs="0" maxOccurs="1" name="CityName" type="s:string"/>
                    <s:element minOccurs="0" maxOccurs="1" name="CountryName" type="s:string"/>
                </s:sequence>
            </s:complexType>
        </s:element>
        <s:element name="GetWeatherResponse">
        <s:element name="GetCitiesByCountry">
        <s:element name="GetCitiesByCountryResponse">
        <s:element name="string" nillable="true" type="s:string"/>
    </s:schema>
</types>
```

Figure 2.3: Structure de l'élément *types* dans la description du service GlobalWeather en WSDL 2.0.

L'élément « `interface` » décrit les opérations du service et la séquence de messages qu'elles envoient et/ou reçoivent. Il regroupe les messages selon l'opération à laquelle ils sont associés. Aussi, une opération est-elle une séquence de "input" et de "output" et une interface est-elle un ensemble d'opérations (cf. figure 2.4).

```
47      <interface name="GlobalWeatherSoap">
48          <operation name="GetWeather"
49          pattern ="http://www.w3.org/ns.wsdl/in-out">
50              <input element="tns:GetWeather"/>
51              <output element="tns/GetWeatherResponse"/>
52          </operation>
53          <operation name="GetCitiesByCountry"
54          pattern ="http://www.w3.org/ns.wsdl/in-out">
58      </interface>
```

Figure 2.4 : Structure de l'élément *interface* dans la description du service GlobalWeather en WSDL 2.0.

L'élément « `binding` » décrit comment accéder au service Web. Il spécifie le protocole de transmission des données pour les opérations et les messages définis au niveau d'une interface donnée (cf. figure 2.5). Dans notre exemple, on remarque que les opérations GetWeather et GetCitiesByCountry utilisent le protocole de communication SOAP.

```
59      <binding xmlns:vsoap="http://www.w3.org/ns/wsdl/soap"
60      name="GlobalWeatherSoap"
61      interface="tns:GlobalWeatherSoap">
62          <operation ref="tns.GetWeather" vsoap:soapAction="http://www.webserviceX.NET/GetWeather"/>
63          <operation ref="tns.GetCitiesByCountry"
64          vsoap:soapAction="http://www.webserviceX.NET/GetCitiesByCountry"/>
65      </binding>
```

Figure 2.5: Structure de l'élément *binding* dans la description du service GlobalWeather en WSDL 2.0.

Pour chaque interface décrite, un élément « `service` » lui est associé (cf. figure 2.6). Celui-ci définit la localisation du service décrit. Le sous-élément `endpoint` spécifie une adresse pour un `binding` par le biais d'un élément `address`. Cette adresse est une URL localisant le service. Ceci permet à une interface d'un service d'avoir plus d'une localisation par le biais de plusieurs éléments `endpoints`. Dans l'exemple *GlobalWeather*, le service est accessible à travers l'URL suivant : "http://www.webservicex.com/globalweather.asmx".

```
66      <service name="GlobalWeather"
67          interface="tns:GlobalWeatherSoap">
68          <endpoint name="GlobalWeatherSoap" binding="tns:GlobalWeatherSoap"
69          adress="http://www.webservicesx.com/globalweather.asmx"/>
70
71      </service>
```

Figure 2.6: Structure de l'élément *service* dans la description du service GlobalWeather en WSDL 2.0.

2.3.2. SAWSDL

SAWSDL (Semantic Annotation for WSDL) s'inscrit dans le cadre des approches à base d'annotations. C'est un langage recommandé par le W3C pour la description sémantique des services Web. Il est évolutif et compatible avec les standards de services Web existants, en particulier avec WSDL [Farell et Lausen, 2007a]. En effet, il augmente l'expressivité du langage WSDL avec la sémantique en utilisant des concepts de l'ontologie.

SAWSDL fournit un mécanisme permettant d'annoter sémantiquement les types de données et les opérations de WSDL. En particulier, il ajoute aussi de nouveaux éléments pour des spécifications spéciales. Pour ce faire, il définit un ensemble d'attributs pour l'extension de WSDL en proposant un mécanisme d'annotation indépendant du langage de représentation sémantique [El Bitar et al., 2013b]. Les annotations sémantiques sont réalisées par le biais des références à des modèles conceptuels comme les ontologies. Au lieu de spécifier un langage pour représenter les modèles sémantiques, SAWSDL prévoit des mécanismes par lesquels les concepts des modèles sémantiques peuvent être référencés à l'aide des annotations. Cela se fait grâce à trois attributs. Le premier «`modelReference`» permet d'associer un composant WSDL ou XML Schema à un concept d'une ontologie. Les deux autres sont les attributs `liftingSchemaMapping` et `loweringSchemaMapping` importants pour le Mapping entre le type de données du concept et celui d'un schémaXML, dans les deux sens.

L'attribut `liftingSchemaMapping` indique le mapping des éléments XML vers un modèle sémantique. Inversement, l'attribut `loweringSchemaMapping` indique le mapping d'un modèle sémantique vers une structure XML (cf. figure 2.7). Ces mises en correspondance sont utiles lors de l'invocation ou la composition de services pour assurer l'échange de données qui n'ont pas la même structure de données.

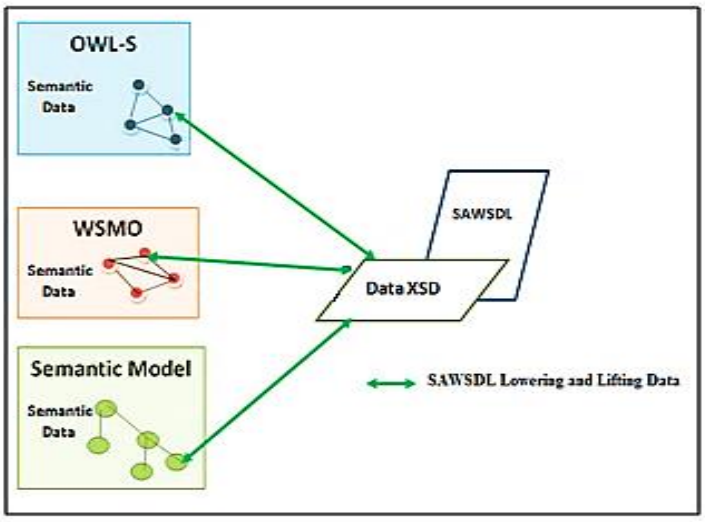


Figure 2.7: Lifting and Lowering Schema de SAWSDL.

Pour illustrer l'utilisation de SAWSDL, nous reprenons l'exemple du service Web *GlobalWeather*. La figure 2.8 présente un extrait de la description SAWSDL de ce service[1].

```
<wsdl:description ... xmlns:sawsdl="http://www.w3.org/ns/sawsdl">
    <rdf:RDF xmlns:base ="http://refapp.semwebcentral.org./weather/ont/weather-ont.owl">
    </rdf:RDF>
        <wsdl:types>
                <s:schema elementFormDefault="qualified" targetNamespace="http://www.webserviceX.NET">
                        <s:element name="GetWeather" ...>
                        </s:element>
                        <s:element name="GetWeatherResponse"
                                sawsdl:modelReference="http://refapp.semwebcentral.org./weather/ont/weather-ont.owl#WeatherObservation">
                                <s:complexType>
                                        <s:sequence>
                                                <s:element minOccurs="0" maxOccurs="1" name="GetWeatherResult" type="s:string" />
                                        </s:sequence>
                                </s:complexType>
                        </s:element>
                        <s:element name="GetCitiesByCountry"
                                sawsdl:loweringSchemaMapping="http://sesa.sti2.at/services/xslt/weather-lowering.xslt">
                                <s:complexType>
                                        <s:sequence>
                                                <s:element minOccurs="0" maxOccurs="1" name="CountryName" type="s:string" />
                                        </s:sequence>
                                </s:complexType>
                        </s:element>
                        ...
                        <s:element name="string" nillable="true" type="s:string" />
                </s:schema>
        </wsdl:types>
        ....
</wsdl:description>
```

Figure 2.8: Extrait de l'annotation sémantique SAWSDL du service GlobalWeather.

[1] http://sesa.sti2.at/services/globalweather.sawsdl

L'élément GetWeatherResponse est une chaîne de caractères qui englobe un ensemble d'informations concernant les observations météorologiques (telles que la température, l'humidité et la vitesse du vent). Il est annoté par le concept WeatherObservation de l'ontologie weather-ont, tandis que l'élément GetCitiesByCountry est annoté par un fichier .xslt indiquant son mapping.

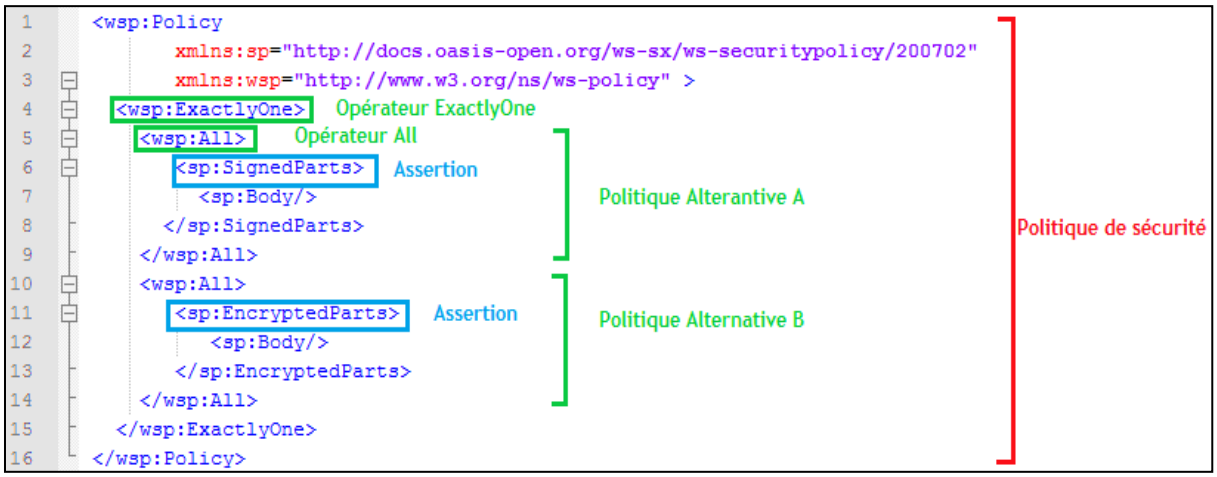
2.3.3. WS-Policy

Les propriétés fonctionnelles constituent le fondement de la description des services Web. Toutefois, ces propriétés ne sont pas suffisantes pour déterminer le service le plus approprié aux besoins spécifiques du client parmi un ensemble de services assurant les mêmes fonctionnalités. D'où l'intérêt d'avoir une description claire de la politique du service en termes de propriétés non-fonctionnelles telles que le niveau de sécurité, le temps de réponse, le prix du service et bien d'autres.

WS-Policy (Web Service Policy) [Vedamuthu et al., 2007], est un standard du W3C basé sur le standard ouvert XML. Il permet de définir des politiques de services Web en termes d'exigences, de préférences et de caractéristiques générales. Il permet, notamment, aux fournisseurs aussi bien que consommateurs de services Web d'exprimer des exigences ou préférences en termes de sécurité ou qualité de service. WS-Policy utilise une grammaire flexible et extensible qui permet d'exprimer les possibilités, les exigences, et les caractéristiques générales des services Web en tant que « policies ». Nous notons que la dernière version de ce standard est la version WS-Policy 1.5.

Conformément à WS-Policy, une politique (*policy*) d'un service Web peut être exprimée par plusieurs politiques alternatives (*Policy Alternative*) dont chacune regroupe un ensemble d'assertions. Chaque assertion (*Assertion*) représente une propriété décrite par le fournisseur ou une préférence souhaitée par le client concernant un sujet donné (Opération, Endpoint, …). WS-Policy utilise deux opérateurs dans l'expression des politiques : l'opérateur *All* et l'opérateur *ExactlyOne*. L'opérateur *All* signifie que toutes les assertions constituant une politique alternative doivent être satisfaites pour accepter cette politique alternative. Par contre l'opérateur *ExactlyOne* signifie qu'au moins une des politiques alternatives doit être satisfaite pour accepter la politique.

Nous illustrons dans la figure 2.9 un extrait de la description WS-Policy relative à la sécurité du message échangé par le service Web *GlobalWeather*. Les assertions *EncryptedParts* et *SignedParts* précisent respectivement les parties du message où est appliquée la politique de sécurité en termes de chiffrement et signature numérique. La politique ci-dessous traduit le fait que le service Web signe ou chiffre le corps du message échangé.




```
<wsp:Policy
        xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
        xmlns:wsp="http://www.w3.org/ns/ws-policy" >
  <wsp:ExactlyOne>
    <wsp:All>
      <sp:SignedParts>
        <sp:Body/>
      </sp:SignedParts>
    </wsp:All>
    <wsp:All>
      <sp:EncryptedParts>
        <sp:Body/>
      </sp:EncryptedParts>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Figure 2.9 : Extrait de la description non fonctionnelle associée au service *GolbalWeather*

Finalement, WS-Policy permet de décrire syntaxiquement les politiques relatives à un service Web relatives à différents domaines non-fonctionnels à savoir la sécurité (WS-PolicySecurity), la qualité de service et bien d'autres.

2.4. Les modèles sémantiques de description

Les modèles sémantiques de description sont fondés essentiellement sur des langages de description des services Web, basés sur les langages particuliers d'ontologie du Web. OWL-S et WSMO sont les plus référencés dans cette catégorie. Nous illustrons dans la suite leurs composants fondamentaux.

2.4.1. OWL-S

OWL-S (Ontologiy Web Language for Service) est une ontologie qui s'inscrit dans le cadre de l'approche à base de langage sémantique. OWL-S permet de décrire les services Web de façon non ambiguë et interprétable par des programmes [Claro et al., 2005] [Lara et al., 2004] [OWL-S Coalition, 2004]. Il est basé sur OWL (Ontology Web Language) [Smith et al., 2004] et constitue une ontologie OWL particulière.

OWL-S couvre un ensemble d'objectifs, rendus possibles par le biais de l'expressivité héritée de OWL et de l'utilisation de la logique de description [Baader et al., 2003], à savoir la

description de services Web sémantiques, leur découverte, leur invocation et leur composition automatiques. Il fournit une spécification des prérequis 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, il définit une ontologie supérieure pour la description, l'invocation et la composition des services Web que nous présentons 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 essentielles pour un service, à savoir : que réalise le service, comment l'utiliser et comment y accéder ?

Que réalise le service ? Cette information est donnée dans le « `Service Profile`». Cette section 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 [OWL-S Coalition, 2004]. Les recherches de services peuvent utiliser n'importe quel élément de « `Profile` » comme critère [Charlet et al., 2003].

Comment utilisons-nous le service ? Cette information est fournie dans le « `Process Model` » qui est utilisé pour décrire essentiellement le fonctionnement (modèle) d'un service composite. OWL-S modélise les services en tant que processus 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 des 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 » et « If-Then-Else ». Un processus simple n'est pas invocable. Il fournit juste une description abstraite d'un processus.

Comment accédons-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. Ce type d'informations est particulièrement utile pour l'invocation automatique de services. La figure 2.10 résume la structure de l'ontologie supérieure de OWL-S. En effet, OWL-S décrit un service à l'aide des trois classes suivantes :

- `ServiceProfile` qui définit le service Web.
- `ProcessModel` pour décrire le fonctionnement du service Web.
- `ServiceGrounding` qui définit comment accéder au service Web.

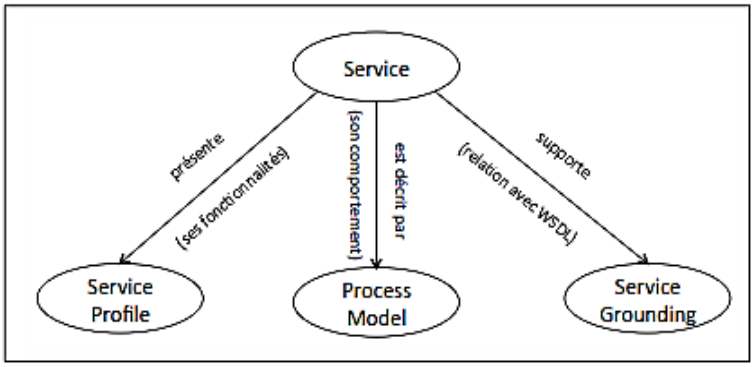


Figure 2.10: Structure de l'ontologie de services OWL-S.

La figure 2.11 présente un extrait de la description OWL-S du service Web GlobalWeather dont nous avons déjà exposé la description en WSDL. Cette description est composée du profil du service (GlobalWeatherProfile), du processus du service (GlobalWeatherProcess) et du grounding du service (WsdlGrounding). Nous l'avons générée au moyen du convertisseur WSDL2OWL-S, proposé par [Paolucci et al., 2003] et disponible sur le site SemWebCentral[2].

```
1  <service: Service rdf:ID="globalWeather">
2      <service:presents rdf:resource="&my_pofile;#GlobalWeatherProfile"/>
3      <service:describedBy rdf:resource="&my_process;#GlobalWeatherProcess"/>
4      <service:supports rdf:resource="&my_grounding;#WsdlGrounding"/>
5  <service:Service>
```

Figure 2.11: Extrait de la description OWL-S du service Web GlobalWeather.

La figure 2.12 présente le profil du service. L'élément « `Profile` » apporte une description du service et de son fournisseur (description abstraite du service). Un identifiant est affecté au profil. Un nom et une description textuelle du service ainsi que les coordonnées et l'adresse du fournisseur à contacter pour plus d'informations sur le service sont aussi spécifiés dans le profil.

Le « `ServiceProfile` » décrit aussi les préconditions fondamentales avant l'appel des opérations et les effets attendus après l'exécution du service demandé. Dans le cas du service

[2] Lien : projects.semwebcentral.org/projects/wsdl2owl-s/

GlobalWeather, ce dernier a des paramètres d'entrée décrits par l'élément `<profile:hasInput>` et des paramètres de sortie décrits par l'élément `<profile:hasOutput>`.

```
<profile:Profile rdf:ID="AdServiceName">
    <profile:serviceName> GlobalWeather </profile:serviceName>
    <profile:textDescription> Get weather report for all major cities around the world. </profile:textDescription>
    <profile:contactInformation>
        <profile:Actor rdf:ID="WebServiceX"/>
    </profile:contactInformation>
    <profile:hasInput>
        <process:Input rdf:ID="GlobalWeatherSoap_GetWeather_parameters_IN">
            <process:parameterType rdf:datatype="&xsd;#anyURI">
                &concept;#getWeatherTypeDeclaration
            </process:parametertype>
        </process:Input>
    </profile:hasInput>
    <profile:hasOutput>
    <profile:hasIntput>
    <profile:hasOutput>
</profile:Profile>
```

Figure 2.12: Extrait du profil du service Web GlobalWeather.

La classe `ServiceModel` décrit le fonctionnement du service Web. Ceci est fait en exprimant les transformations faites par le service Web sur les données (input à output), et les transformations d'état (préconditions et effets).

Les services Web peuvent être modélisés avec OWL-S en tant que processus grâce à la classe « Process ». Un processus est une spécification de la manière dont le client peut interagir avec le service. Il est composé d'un ensemble d'informations sur ce service : son nom, les participants à son exécution (un client simple ou d'autres services Web), ce qu'il fait, les conditions de son utilisation et ses effets, son résultat et ses paramètres (entrée, sortie).
Dans notre travail, nous nous intéressons à la description d'un service Web élémentaire (atomique), ainsi nous présentons ci-après la description d'un processus atomique.

Un processus atomique est un processus exécutable en une seule étape. Son exécution correspond à une unique avancée dans l'exécution du service, il est directement invoqué par l'utilisateur du service suivant les critères du `GroundingProfile`. Le service *GlobalWeather* peut être décrit par deux processus atomiques

GlobalWeatherSoap_GetWeather et GlobalWeatherSoap_GetCitiesByCountry tout en spécifiant leurs paramètres d'entrée et sortie (cf. figure 2.13).

```
<process:Input rdf:ID="GlobalWeatherSoap_GetWeather_parameters-IN">
...
</process:Input>
<process:Output rdf:ID="GlobalWeatherSoap_GetWeather_parameters_OUT">
...
</process:Output>
<process:AtomicProcess rdf:ID="GlobalWeatherSoap_GetWeather">
    <process:hasInput rdf:resource="#GlobalWeatherSoap_GetWeather_parameters-IN"/>
    <process:hasOutput rdf:resource="#GlobalWeatherSoap_GetWeather_parameters_OUT"/>
</process:AtomicProcess>

<process:Input rdf:ID="GlobalWeatherSoap_GetCitiesByCountry_parameters-IN">
<process:Output rdf:ID="GlobalWeatherSoap_GetCitiesByCountry_parameters_OUT">
<process:AtomicProcess rdf:ID="GlobalWeatherSoap_GetCitiesByCountry">
```

Figure 2.13 : Processus atomiques du service Web GlobalWeather.

2.4.2. WSMO

WSMO (Web Service Modeling Ontology) est une ontologie qui décrit les différents aspects de la composition dynamique de services Web, y compris la découverte dynamique, la sélection, la médiation et l'invocation. Il est basé sur WSMF (Web Service Modelling Framework) [Fensel et al., 2002] qui précise les principaux éléments de description sémantique des services Web. Par ailleurs, pour modéliser un service Web, WSMO utilise le langage WSML (Web Service Modeling Language) [Bruijin et al., 2005]. WSMO propose quatre éléments clés pour modéliser les différents aspects des services Web sémantiques: les ontologies, les médiateurs, les objectifs ou goals, et les services.

WSMO Ontologies : Les ontologies fournissent une terminologie pour décrire sémantiquement des éléments appartenant à des domaines spécifiques en fournissant les concepts et les relations entre eux. Elles regroupent un ensmeble d'atributs à savoir : concepts, relations, fonctions, instances et axiomes. Afin de décrire les propriétés sémantiques des relations et des concepts, une ontologie WSMO fournit aussi un ensemble d'axiomes, qui sont exprimés dans un langage logique.

WSMO Mediators : WSMO utilise un ensemble de médiateurs pour résoudre les incompatibilités (structurelles, sémantiques ou conceptuelles) détectées au niveau données ou processus afin de connecter les ressources WSMO hétérogènes. On en distingue quatre types :

- ggMediateurs : médiateurs qui connectent deux buts (goal to goal).
- ooMédiateurs : médiateurs qui importent des ontologies et permettent de résoudre l'inadéquation de représentation possibles entre ontologies. (ontology to ontology)
- wgMédiateurs : médiateurs qui connectent les services Web à des objectifs. (Web service to goal)
- wwMédiateurs: médiateurs qui connectent deux services Web. (Web service to Web service)

WSMO goals : WSMO décrit les buts qu'un utilisateur de service Web cherche à satisfaire à travers le *goal*. L'utilisateur définit ses critères de recherche en décrivant l'interface et les fonctionnalités attendues à travers la description du but. Un médiateur spécifique est utilisé pour connecter le but demandé avec le service Web correspondant.
Un but peut importer des ontologies en utilisant le médiateur d'ontologie (ooMediator) si jamais des problèmes d'alignement et de fusion des ontologies importées sont relevés. Un but peut être défini en réutilisant un ou plusieurs objectifs existants. Ceci est réalisé à l'aide du médiateur de but (ggMediator).

WSMO Web services : Différents aspects sont considérés dans la description d'un service Web WSMO, à savoir l'aspect fonctionnel et l'aspect comportemental. Pour ce faire, WSMO utilise deux points de vue différents : la capacité (*Capability*) et l'interface (*Interface*). La capacité renseigne sur les propriétés fonctionnelles du service en décrivant les variables partagées (*SharedVariables*) entre les préconditions (*Preconditions*), postconditions (*Postconditions*), assomptions (*Assumptions*) et effets (*Effects*).
L'interface décrit la façon dont la fonctionnalité du service Web peut être réalisé en spécifiant son comportement (orchestration et chorégraphie) vis-à-vis de ses partenaires. Aussi, un ensemble optionnel de propriétés prédéfinies (comme le propriétaire, l'éditeur, la version, etc.) peut-il être associé à un service WSMO.
La figure 2.14 illustre l'exemple du service Web *GlobalWeather* décrit dans WSMO au moyen du langage WSML.

```
WebService
Capability GlobalWeather

precondition
            definedBy
                        ?CityName memberOf po#OntologyCountry
                    and     po#ExistCity (?CityName)

Assumption
            definedBy
            po#ValidDate(?Date)

Postconditon
            definedBy
            ?Forecast memberOf tr#OntologyForecast

Effect
            definedBy
            po#ShowForecast(?Cityname ?Date)
```

Figure 2.14 : Extrait du service Web GlobalWeather décrit en WSML dans WSMO

2.5. Etude des approches de description de services Web

Une description enrichie d'un service Web doit inclure des aspects clés [Sheth, 2003], à savoir l'aspect fonctionnel et l'aspect non-fonctionnel. Chacun de ces aspects fournit un type très spécifique d'information pour la description de service :

- L'aspect fonctionnel fournit des informations sur ce que le service peut fournir à ses clients en termes de fonctionnalités, lors de son invocation.
- L'aspect non-fonctionnel, quant à lui, fournit des informations sur tous les paramètres décrivant le comportement ou l'utilisation du service tels que la qualité de service, la fiabilité, la performance, la sécurité, ou encore le prix [Chung et al., 1999]. Les propriétés non-fonctionnelles d'un service Web sont souvent désignées comme des contraintes ou des préférences qui reflètent les attributs de qualité, ou tout simplement les paramètres de service qui ne renseignent pas sur sa fonctionnalité.

Toutefois, les approches de description de services Web sémantiques présentées auparavant s'intéressent strictement à l'aspect fonctionnel du service.

Très peu de travaux se sont préoccupés de l'aspect non-fonctionnel. Cependant, malgré l'enrichissement qu'ils apportent à la description non-fonctionnelle du service Web, ces travaux se limitent au niveau syntaxique.

Dans la suite, nous exposons notre étude des approches de description de services Web par rapport aux aspects fonctionnel et non-fonctionnel.

2.5.1. Aspect fonctionnel

Généralement, l'aspect fonctionnel d'un service couvre les opérations qu'il offre et la manière de les appeler [Klusch, 2008]. Dans la panoplie d'approches couvrant cet aspect, nous limitons notre étude aux candidats les plus connus OWL-S, WSMO et WSDL/SAWSDL.

Le *ServiceProfile* de OWL-S [Martin et al., 2004] décrit la signature du service et ses capacités par l'intermédiaire de quatre éléments qui sont définis par : les paramètres d'entrées (*inputs*), les paramètres de sorties (*outputs*), les préconditions (*preconditions*), et les effets (*effects*).
Par ailleurs, WSMO décrit l'aspect fonctionnel du service Web, à travers les attributs buts (*Goals*) et capacités du service (*WS capabilities*) [Bruijn et al., 2005]. Il utilise des conditions supplémentaires sur les entrées et les sorties du service Web ceci, en définissant les préconditions (*preconditions*), les postconditions (*postconditions)*, les hypothèses (*assumptions*) et les effets (*effects*). En ce qui concerne les erreurs survenues durant l'échange d'un message contenant une référence de service Web, les éléments *Fault* et *endpoint* spécifient le récepteur des erreurs liés à ce message.

En ce qui concerne les standards du consortium W3C, deux langages sont concernés par la description fonctionnelle des services Web, WSDL 2.0 et SAWSDL. WSDL 2.0 [Chinnici et al., 2007], limité par sa description syntaxique, définit la description fonctionnelle de service Web par les paramètres : interface (*interface*), opérations (*operations*), paramètres d'entrée (*inputs*) et de sortie (*outputs*), et erreurs (*faults*). Toutefois, le manque de sémantique dans WSDL 2.0 n'est plus un problème. SAWSDL définit trois attributs qui étendent WSDL 2.0 afin d'annoter les services Web sémantiquement :

i) modelReference pour spécifier les associations entre les composants WSDL et les concepts sémantiques,
ii) *LiftingSchemaMapping* et *LoweringSchemaMapping* pour spécifier les mappings entre les données sémantiques et XML.

En plus, SAWSDL ne nécessite aucun changement ou adaptation des documents WSDL existants. Pour mieux comprendre comment chacun des candidats étudiés vise à décrire l'aspect fonctionnel d'un service Web, nous exposons, dans le tableau 2.1, la capacité de chaque candidat à couvrir explicitement les différents éléments décrivant la fonctionnalité d'un service Web présentés ci-après :

- *Catégorie de service :* Elle fait référence à un domaine métier.
- *Fonctionnalité :* Elle décrit l'objectif du service.
- *Input / Output* : Ils correspondent aux paramètres d'entrée et de sortie.
- *Préconditions, Postconditions, Assomptions et Effets*: Ils correspondent à des contraintes à vérifier avant ou après l'exécution du service.
- *Fault (Exception):* Il fait référence aux messages d'erreur.

	OWL-S	**WSMO**	**WSDL/SAWSDL**
Catégorie de service	ServiceCategory	Ontology Hiearchy	Interface/ModelRefernce of Interface
Fonctionnalité	Service: presents	Web service capability identifier	Operation/ModelReference of operation
Input	HasInput	Precondition	Input/ModelReference of input
Output	HasOntput	Postcondition	Ontput/ModelReference of Ontput
Préconditions	HasPreconditions	Precondition	-
Postcondition	-	Postconditon	-
Assomptions	-	Assumptions	-
Effets	HasResult	Effect	-
Fault	HasPrecondition	Precondition	Fault/ModelRefernce of Fault

Tableau 2.1 : Couverture des éléments de description fonctionnelle d'un service Web.

2.5.2. Aspect non-fonctionnel

En ce qui concerne l'aspect non-fonctionnel des services Web, le standard WS-Policy fournit un formalisme pour décrire leurs propriétés non-fonctionnelles.

Il propose une grammaire syntaxique simple et extensible pour décrire les politiques d'un service Web. Ce standard est basé sur XML. En utilisant la grammaire WS-Policy, chaque fournisseur de services, peut définir des assertions relatives à des domaines non-fonctionnels génériques, tels que la sécurité et la qualité de service à travers des spécifications de service Web existantes (*WS-* specifications*) (par exemple, *WS-SecurityPolicy* [Nadalin et al., 2009] et *WS-Reliability* [OASIS, 2004] , etc), ou bien des assertions spécifiques à son contexte.
Par ailleurs, dans WSDL 2.0, les politiques peuvent être liées aux éléments *Binding, Binding operation, input, output, Service* et aussi *Endpoint*.

Les deux ontologies OWL-S et WSMO, quant à elles, mettent l'accent sur la sémantique des services Web et non pas leurs propriétés non-fonctionnelles. Des travaux de recherche ont été présentés dans l'objectif d'étendre ces langages et couvrir les propriétés de qualité de services [Toma et al., 2006] [Kritikos, 2005]. Ces travaux utilisent un paramètre particulier pour intégrer des concepts des propriétés non-fonctionnelles. Néanmoins, ce paramètre prévu pour permettre de définir de nouvelles propriétés de services Web non prévues par les modèles sémantiques OWL-S et WSMO, ne fournit pas une structure adaptée et standardisée pour la description des propriétés non-fonctionnelles des services Web.

2.5.3. Synthèse

L'étude comparative présentée ci-avant, qui a fait l'objet d'un article de synthèse [El Bitar et al., 2013a], nous a permis de dévoiler les mécanismes particuliers de description adoptés par chacun des langages étudiés. A cet égard, nous constatons que les standards W3C se distinguent par la simplicité de la description qu'ils proposent. OWL-S et WSMO, quant à eux, se voient gratifier de nouvelles normes de description de services Web sémantiques qui ont été proposées pour concurrencer le standard WSDL.

Toutefois, ces deux modèles utilisent des structures de propriétés descriptives différentes pour la description des services Web. A titre d'exemples, les inputs et les outputs dans OWL-S sont considérés respectivement des préconditions et postconditions dans WSMO et les éléments du *Service Profile* de OWL-S diffèrent de ceux définis au niveau de *Web Services Capability* dans WSMO. Cette hétérogénéité rend l'appariement d'un service décrit par OWL-S avec un service décrit par WSMO une opération complexe voire impossible vu que certaines structures dans l'un de ces modèles n'ont pas leurs correspondants dans l'autre. D'où l'utilité d'opter pour une description normalisée qui permet d'avoir des structures unifiées à comparer. Notamment, WSDL et SAWSDL, qui sont des standards pour la description des services Web, et leur annotation sémantique sont utilisés en pratique à l'encontre de OWL-S et WSMO qui restent confinés dans les laboratoires.

Ajoutons que OWL-S et WSMO permettent d'annoter un service Web par des ontologies OWL (pour OWL-S) et WSML (pour WSMO), contrairement à SAWSDL, qui reste ouvert à différentes ontologies. En plus, la description des services Web proposée par ces deux modèles reste une description faite à un niveau abstrait qui fait référence aux fichiers WSDL

afin d'obtenir les détails techniques nécessaires pour l'invocation des services. En outre, ces deux modèles ne sont pas adaptés pour la description des propriétés non-fonctionnelles des services Web. Ils mettent l'accent sur leur aspect sémantique et ne fournissent pas une structure standardisée et adaptée pour l'intégration de propriétés non-fonctionnelles de manière à distinguer les politiques offertes au niveau des différents points d'accès d'un service Web.
A l'encontre, les standards W3C incluent le standard de WS-Policy permettant d'attacher des politiques à différents éléments du fichier WSDL.

2.6. Conclusion

Dans ce chapitre, nous avons étudié les langages les plus utilisés dans la littérature pour décrire les services Web. Cette étape de description est le premier processus indispensable (après l'implémentation du service) dans le cycle de vie d'une application basée sur une architecture SOA et est un fondement de base pour la découverte, la sélection et ainsi la composition de services web.

Le langage WSDL est un standard W3C utilisé comme langage de base dans l'industrie pour la description des services Web implémentant des architectures du SOA. Etant donné qu'il se limite à la description syntaxique des services, de nombreux travaux ont été proposés dans l'objectif d'intégrer la sémantique des services Web, entre autres, nous citons les approches les plus connues telles que OWL-S, WSMO et SAWSDL.

Étant proposé pour étendre WSDL, SAWSDL nécessite peu d'efforts de la part des développeurs habitués à WSDL contrairement à OWL-S et WSMO. En plus, SAWSDL est ouvert à tout type d'ontologies tandis que OWL-S et WSMO n'acceptent que des ontologies spécifiques OWL (pour OWL-S) et WSML (pour WSMO). Notons que ces deux modèles proposent des structures différentes pour la description des services Web. Pour le cas de OWL-S qui reste très utilisé dans la littérature, les outils de découverte proposés tels que OWLS-Matcher [Jaeger, 2011], implémentent des algorithmes d'appariement sémantique couvrant uniquement des services annotés suivant l'ontologie OWL-S.

Nous notons également que OWL-S et WSMO font recours à WSDL pour apporter les informations d'ordre technique nécessaires pour invoquer un service Web et ne sont pas dédiés à la description des propriétés non-fonctionnelles des services Web.
Par ailleurs, SAWSDL est un standard d'annotations sémantiques des éléments du langage de base WSDL qui est un langage extensible. Cette caractéristique de WSDL permet d'y inclure facilement et simplement de nouveaux éléments quand cela est nécessaire dans un but d'obtenir une description assez étoffée des services Web. En plus, ce standard peut être combiné au standard WS-Policy pour permettre de décrire les propriétés non-fonctionnelles des services Web. C'est enfin aux standards WSDL, SAWSDL et WS-Policy que nous nous intéressons dans notre travail.

Une description étoffée des services Web est un élément clé pour l'automatisation de la découverte et de l'appariement de services. Plusieurs approches de découverte de service Web ont été formalisées dans la littérature. Nous en discuteront dans le chapitre qui suit.

Chapitre 3

Approches de découverte de services Web sémantiques

SOMMAIRE :

3.1. Introduction

Les services Web ont marqué l'ingénierie du Web par la création d'un framework universel qui exploite les protocoles Internet existants et les standards ouverts XML pour favoriser l'interaction B2B [Christos et al., 2006]. Autrement dit, ils constituent des composants logiciels interopérables pouvant être réutilisés dans le développement d'applications à base de composants et permettant leur intégration.

Dans ce contexte, assister le concepteur ou le développeur dans la recherche de composants nécessaires contribuera à diminuer le coût de développement de nouvelles applications. Aussi, plusieurs approches de découverte de services Web ont-elles été proposées dans la littérature. En parallèle, le contexte du Web a beaucoup évolué grâce à l'évolution du Web sémantique et à l'adoption croissante des services Web comme technologie d'implémentation des services, et par la suite l'augmentation accélérée du nombre des services Web sur Internet. S'ajoute à cela, l'évolution des contraintes des clients qui sont devenus plus exigeants et cherchent à réutiliser des services, qui non seulement répondent à leurs besoins d'ordre fonctionnel, mais également à leurs besoins non-fonctionnels.

Conformément au paradigme des services Web, leurs descriptions sont publiées dans des registres spécialement conçus à cet effet (par exemple, des annuaires *Universal Description Discovery and Integration* ou UDDI). Le but de ces registres est de faciliter la recherche des services Web publiés, aux différents organismes commerciaux ou simples consommateurs souhaitant utiliser un service donné. Localiser un service Web d'intérêt particulier à partir d'un *pool* de services disponibles est la tâche fondamentale de toute approche de découverte de services Web [Savitha, 2009]. Toutefois, des mécanismes demeurent nécessaires pour garantir une sélection efficace de l'instance du service Web approprié en termes de facteurs de qualité et de performance au moment de la consommation du service Web [Patil et Gopal, 2011].

Communément, les procédures de découverte de services Web sont fondées sur une recherche par mots clés et guidées par une intervention manuelle. McCabe et ses co-auteurs [McCabe et al., 2003], ont défini le mécanisme de découverte comme étant "*l'acte de localisation d'une description, traitable par machine, d'un service Web non connu auparavant décrivant certains critères fonctionnels*".

Ainsi, en réponse aux besoins signalés dans sa requête, le client reçoit une liste de descriptions de services Web qu'il devrait encore parcourir manuellement pour en sélectionner les services qui répondent exactement à ses besoins. Toutefois, dans un environnement d'intégration dynamique de systèmes distribués, une recherche rapide, sémantique et automatique des services Web composables est recommandée. La découverte automatique de services Web cherche à identifier des services qui peuvent répondre à une requête donnée en procédant à un appariement (matching) des éléments de la requête et ceux des services décrits dans un annuaire (cf. figure 3.1). Ce mécanisme consiste généralement à identifier un degré de similitude entre les différents concepts sémantiques qui décrivent le service requis (la requête) et ceux des services offerts (les services publiés). Dans le cas des services décrits au moyen du standard W3C WSDL (Web Service Description Language), l'appariement porte sur un ensemble d'éléments tels que les `inputs` et les `outputs`.

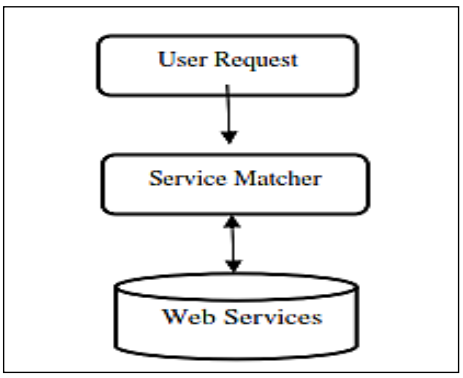


Figure 3.1 : Découverte de services Web

Bien que le domaine de découverte de services Web soit assez récent, beaucoup de travaux ont été consacrés à cet axe. Ils ont été classés dans la littérature selon des critères d'architecture (centralisée, distribuée ou hybride), d'organisation (structurelle ou non-structurelle) et de niveau de matching (syntaxique ou sémantique). Cependant, nous constatons que ces classifications sont obsolètes et portent sur des critères secondaires. Nous pensons que ces travaux doivent davantage être classés selon le formalisme et par la suite, le raisonnement appliqué pour calculer le degré de matching qui constitue un critère primaire, puisque l'efficacité du processus de découverte dépendra essentiellement de la manière dont le matching est effectué. Une telle classification des travaux existants facilitera l'évaluation de leur pertinence. C'est dans ce sens que s'inscrit le travail présenté dans ce chapitre. Nous y présentons d'abord une classification des travaux sur la découverte des services Web avant d'évaluer leur performance en dressant des critères en phase avec les nouvelles exigences des

clients et l'évolution des technologies du Web et des normes de description des services Web sémantiques. Nous avons privilégié les services Web sémantiques vu le contexte actuel du Web qui s'est développé en Web sémantique et l'impact d'un traitement sémantique sur l'efficience de tout processus automatique.

Ce chapitre vise donc à analyser les approches proposées dans la littérature en vue d'identifier des mécanismes qui constituent des points forts de ces approches, et qui par la suite, serviront de guideline pour déterminer les fondements d'une approche de découverte adaptée au contexte actuel des services Web. Après une étude des principales catégories d'approches de découverte de services Web, l'accent sera mis en particulier sur les approches par le raisonnement à partir de cas qui cadrent étroitement avec notre travail.

3.2. Approches de découverte de services Web sémantiques

Notre analyse des travaux consacrés à la découverte des services Web nous a permis de constater que les approches adoptées présentent des hétérogénéités au niveau de la méthode de matching utilisée. Selon ce critère, elles peuvent être regroupées en trois classes : l'approche algébrique, l'approche déductive et l'approche hybride. L'approche algébrique exploite généralement la théorie des graphes et procède par calcul de distance, tandis que l'approche de découverte déductive est fondée essentiellement sur la logique. L'approche qualifiée d'hybride, quant à elle, est celle qui combine les mécanismes de ces deux classes.

La classification que nous avons adoptée est similaire à celle proposée dans les travaux de Schumacher et al. [Schumacher et al., 2008] et Chabeb [Chabeb, 2011] qui ont recensé et étudié trois catégories d'approches de découverte de services Web: logiques, non-logiques et hybrides. Cependant, nous notons que leurs études n'ont pas intégré une catégorie importante des approches hybrides qui est celles des travaux intelligents à base de RàPC.

Dans la suite, nous présentons les principes de chacune de ces approches et un ensemble de travaux ou de plateformes qui l'adoptent. Nous présentons dans la première partie de ce chapitre les approches de découverte classiques et nous les étudions sur la base des critères de comparaison que nous avons identifiés. La deuxième partie du chapitre sera consacrée à la présentation et à l'étude des travaux de découverte intelligente à base de RàPC qui sont connexes à notre travail.

3.2.1. Approche algébrique

L'approche de découverte algébrique trouve ses origines dans l'algèbre et les mécanismes de recherche d'information. Elle est fondée sur le calcul du degré de similarité textuelle à partir de graphes structurés construits à cet effet, ou encore sur le calcul de distance (du chemin) entre les concepts appariés. Cette approche que nous qualifions d'algébrique utilise des mécanismes de matching structurel, numérique et syntaxique à travers un appariement de graphes structurés et en calculant des distances numériques pour vérifier la similitude syntaxique. Pour exploiter la sémantique, ces mécanismes d'appariement utilisent les fréquences des termes et les sous-graphes. iMatcher1 [Schumacher et al., 2008], Agent Approach for Service Discovery and Utilization (AASDU) [Palathingal et Chandra, 2004] et DSD-Matchmaker [Klein et König-ries, 2004a] sont des plateformes et travaux qui adoptent cette catégorie d'approches.

3.2.1.1 iMatcher1

iMatcher1 [Schumacher et al., 2008] est un système de découverte non logique de services Web qui utilise un matchmaker syntaxique de profils de services. Ce système puise des services à partir d'un ensemble de profils de services Web décrits en OWL-S. Ces services sont stockés sous forme de graphes RDF (Resource Description Framework) sérialisés dans une base de données RDF, au moyen d'une extension du langage RDQL (RDF Data Query Language), appelée iRDQL [Bernstein et Kiefer, 2005]. Le degré d'appariement de la requête et d'un service est calculé à partir de quatre métriques de calcul de similarité syntaxique : TFIDF (Term Frequency-Inverse Document Frequency) [Jones, 1972] [Salton, 1983], la distance de similarité de Levenshtein [Levenshtein, 1966] [Gusfield, 1997], la mesure du vecteur Cosinus [Garcia, 2006] [Zhu et al., 2010] et la mesure de divergence de Jensen-Shannon [Fuglede et Topsoe, 2004]. Les résultats sont classés en fonction des scores numériques de ces mesures de similarités syntaxiques et d'un seuil défini par l'utilisateur.

3.2.1.2 AASDU

L'approche AASDU (Agent Approach for Service Discovery and Utilization) est une approche multi-agents proposée pour la découverte de services Web [Palathingal et Chandra, 2004].

Le système AASDU contient quatre composants : une interface utilisateur graphique (Graphical User Interface GUI), un agent analyseur de requête (Query Analyzer Agent QAA),

un système référentiel des domaines d'expertises des agents de service et un module de services.

Le système référentiel des domaines d'expertises des agents de service permet de référencer les agents selon leur expertise, ainsi il n'est pas nécessaire que chaque agent ait connaissance de tous les services publiés dans les registres distribués. Chaque agent a juste connaissance des services relatifs à son domaine d'expertise. Dans ce système, chaque agent a un profil déterminant ses intérêts et son expertise. L'expertise de l'agent est représentée par un vecteur de mots-clés tel que chaque mot clé représente un domaine donné. Pour chaque mot-clé, un score est assigné indiquant le degré d'expertise de l'agent dans ce domaine. De plus, chaque agent a une liste d'agents voisins (Neighbor list) qui indique les domaines d'expertise de ses voisins. Lorsqu'un agent vient rejoindre le système, un ensemble de voisins lui est assigné de façon aléatoire.

Le composant module de services offre trois sous-services. Le premier sous-service permet aux fournisseurs de services de publier les descriptions de leurs services Web. Pour chaque fournisseur de service Web, un agent lui est assigné. Le deuxième sous-service consiste en un agent de négociation permettant la sélection de service. Le troisième sous-service est offert par l'agent composition dont le rôle est d'invoquer l'un des services issus de l'étape de sélection ou d'invoquer un service similaire lors de la défaillance du service sélectionné en premier.

Selon cette approche, la requête est exprimée sous forme d'une chaîne de caractères, via une interface GUI (Graphical User Interface). Elle est ensuite envoyée à un agent Query Analyzer Agent (QAA). Ce dernier en extrait les mots-clés pertinents qu'il utilise par la suite, pour sélectionner d'autres agents experts à partir du système référentiel des domaines d'expertise des agents service. Pour ce faire, il réalise un appariement syntaxique fondé sur une simple variante de la technique TFIDF (Term Frenquency Inverse Document Frequency) [Letsche T.A. et Berry, 1997]. Les agents experts sélectionnés transmettent ensuite les paramètres des services appartenant à leur domaine d'expertise, à un agent de composition. Ce dernier invoque l'un des services candidats selon le choix de l'utilisateur ou compose certains parmi eux pour répondre à sa requête.

3.2.1.3 DSD-matchmaker

DSD-matchmaker [Klein et König-ries, 2004 a] réalise la découverte de services Web à travers l'appariement de graphes de description de services. Ces descriptions sont spécifiées au moyen du langage orienté objet de description de service DSD (Diane Service Description) [Kuster et Konig-Ries, 2007] 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 à partir d'un graphe les variables à satisfaire, et sélectionne, en se basant sur l'état des services, celui qui répond le mieux à la requête parmi l'ensemble de services découverts, puis il renvoie une valeur numérique représentant le degré d'appariement correspondant.

Le fonctionnement du DSD-matchmaker se fait en plusieurs étapes au cours desquelles il effectue des estimations des offres de services. Il détermine d'abord 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 du service proposé est prometteuse, c'est-à-dire si la valeur fournie par cette opération est utile pour décider à quel niveau ce service répond à la requête initiale.
Dans une deuxième étape, le DSD-matchmaker n'exécute que 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 est effectué sur la base des descriptions mises à jour. Dans [Klusch et al., 2008], les auteurs expliquent leur approche et proposent d'intégrer des informations dynamiques dans les descriptions des services.

Pour effectuer le processus d'appariement, 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 [Küster et al., 2007]. L'algorithme [Klein et König-ries, 2004a] [Klein et Konig-ries, 2004b] [Klein et al., 2005] utilisé pour ce faire calcule le degré d'appariement de deux nœuds en tant qu'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.

3.2.2. Approche déductive

L'approche déductive est fondée sur la logique. Les travaux optant pour cette catégorie d'approches utilisent des descriptions de services et des requêtes spécifiées en langages issus de formalismes logiques, telles que la logique de description et la logique de premier ordre. Ils utilisent également des règles logiques pour la découverte de services Web et exploitent les ontologies pour couvrir leur aspect sémantique. Pour calculer le degré d'appariement, ils recourent à différentes façons et ciblent plusieurs éléments de description de services tout en prenant en considération leur sémantique. Ils optent essentiellement pour trois types d'appariement : IO-matching (Inputs and Outputs matching) [Srinivasan et al., 2004a] [Fan et al., 2005] [Paolucci et al., 2002], PE-matching (preconditions and effects matching) [Schumacher et al., 2008] et IOPE-matching (Inputs, Outputs, preconditions and effects matching) [Jaeger et al., 2005] [Keller et al., 2005] [Stollberg et al., 2007] [Küster et Konig-Ries, 2008]. Dans l'IO-matching, les éléments objet d'appariement se limitent aux inputs et outputs. PE-matching prend les préconditions et effets comme éléments d'appariement. Quant au cas du IOPE-matching, les concepts sémantiques décrivant les inputs, outputs, préconditions et effets font à la fois l'objet de l'appariement des services et des requêtes.

3.2.2.1 WSC

Paolucci et ses co-auteurs ont défini l'approche WSC (Web Services Capabilities) [Paolucci et al., 2002] qui propose un appariement sémantique des capacités des services Web fondé sur l'utilisation de l'ontologie DAML [DARPA, 2003]. Les services publiés et les requêtes font référence à des concepts DAML qui les décrivent sémantiquement. L'appariement adopté est de type IO-matching. En appariant des concepts sémantiques de type DAML, un service publié est considéré comme un service qui répond bien 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. Quatre degrés d'appariement sont proposés : EXACT, PLUGIN, SUBSUMES et FAIL [Paolucci et al., 2002].

La figure 3.2 illustre l'algorithme des règles d'affectation des degrés d'appariement des outputs. Considérons *Serv* un service publié, *Q* une requête de service, et *outServ* et *outQ* deux de leurs outputs respectives.

L'algorithme précise que le degré EXACT est le résultat d'appariement des deux outputs de *Serv* et de *Q*, lorsque ceux-ci sont équivalents ou lorsque l'output de Q est une sous classe du output de *Serv*. Le degré PLUGIN correspond au cas où l'output de *Serv* est un concept plus

générique que l'output de *Q*. Le degré SUBSUMES correspond au cas où l'output de *Q* est plus générique que l'output de *Serv*. Quant au degré FAIL, il correspond au cas où aucune relation hiérarchique entre les deux outputs à apparier n'est identifiée.

```
degreeOfMatch (outQ, outServ):
    if outServ=outQ then return exact
    if outQ subclassOf outServ then return exact
    if outServ subsumes outQ then return plugIn
    if outQ subsumes outServ then return subsumes
    otherwise fail
```

Figure 3.2 : Règles d'affectation des degrés d'appariement d'outputs dans WSC. Source [Paolucci et al., 2002].

Par ailleurs, le processus de sélection des services Web découverts dans WSC se base sur la sélection du service possédant le meilleur score au niveau des outputs. La comparaison des scores d'appariement des inputs obtenus pour chaque service, n'est en fait effectuée que dans le cas où les scores d'appariement des outputs de ces services sont égaux. La figure 3.3 présente l'algorithme de tri des appariements de deux services dans WSC dans le but d'en sélectionner le meilleur. Soient deux services offerts Serv1 et Serv2, match1 et match2 réfèrent aux matching de chacun des services avec une requête donnée. L'algorithme trie les deux appariements dénotés *match i* comme suit :

- Si le score d'appariement des outputs du premier service est plus grand que le score d'appariement des outputs du deuxième service alors c'est le premier service qui sera sélectionné.
- Dans le cas où le score d'appariement des outputs du premier service est égal au score d'appariement des outputs du deuxième service, si le score d'appariement des inputs du premier service est plus grand que le score d'appariement des inputs du deuxième alors c'est le premier service qui sera sélectionné.
- Dans le cas où le score d'appariement des outputs du premier service est égal au score d'appariement des outputs du deuxième service, si le score d'appariement des inputs du premier service est égal au score d'appariement des inputs du deuxième alors les deux services seront sélectionnés.

```
sortRule(match1, match2){
    if (match1.output > match2.output) then match1 > match2
    if ((match1.output = match2.output)
        & (match1.input > match2.input)) then match1 > match2
    if ((match1.output = match2.output)
        & (match1.input = match2.input)) then match1 = match2
}
```

Figure 3.3 : Algorithme de tri des appariements des services dans WSC. [Source : Paolucci et al., 2002].

3.2.2.2 OWLS-M

Jaeger et ses co-auteurs proposent un matchmaker OWL-S fondé sur le IOPE-matching [Jaeger et al., 2005]. Ce matchmaker recherche des correspondances sémantiques entre les paramètres fonctionnels définis dans des descriptions de services Web OWL-S et ceux introduits dans la requête [Srinivasan et al., 2004a] [Srinivasan et al., 2004b] [Srinivasan et al., 2006].

Le processus d'appariement global comporte quatre étapes de base: l'appariement des inputs, l'appariement des outputs, l'appariement des catégories de services et une quatrième tâche pendant laquelle des contraintes prédéfinies par l'utilisateur sont prises en considération. L'algorithme calcule dans chacune des trois premières étapes le degré d'appariement et effectue la quatrième tâche pour vérifier les conditions définies par l'utilisateur. Ensuite, OWLS-M passe à l'agrégation des résultats d'appariement obtenus et retourne le rang du service Web apparié s'il est bien retenu comme service répondant à la requête. Ce rang classe le service selon son degré d'appariement parmi l'ensemble des services Web découverts. La figure 3.4 illustre le processus d'agrégation des appariements dans OWLS-M.

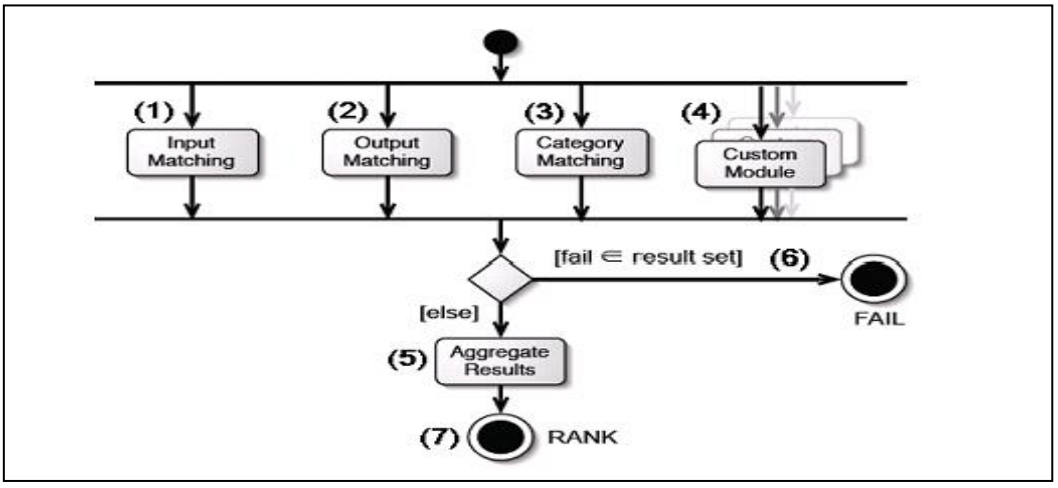


Figure 3.4 : Agrégation des degrés d'appariement dans OWLS-M. [Source : Jaeger et al., 2005].

Les auteurs proposent quatre types de degrés d'appariement :

- 0 ou FAIL si l'input (l'output) requis n'a pas de correspondant dans le service,
- 1 ou UNKNOWN si l'algorithme d'appariement ne peut identifier la catégorie d'un input (output),
- 2 ou SUBSUMES si l'input (l'output) offert est plus spécifique que l'input (l'output) requis,
- 3 ou EQUIVALENT en cas d'équivalence entre inputs (outputs) appariés.

Ces mêmes degrés sont appliqués pour l'appariement des catégories de services.

3.2.2.3 ALS

Keller et ses co-auteurs proposent un modèle de localisation automatique et sémantique de services (Automatic Location of Services, ALS) [Keller et al., 2005] (cf. figure 3.5). Ce modèle utilise des objectifs (*goals*) prédéfinis et des abstractions de services pour enfin découvrir des services concrets répondant à une requête.

Le processus de localisation commence par la formulation du besoin de l'utilisateur (en termes d'outputs et d'effets) pour sélectionner auprès du registre d'objectifs prédéfinis (*Goal-Repos*) l'objectif (*goal*) correspondant à la requête de l'utilisateur (cf. figure 3.5, partie (a)). Ensuite, l'objectif sélectionné dans la première phase sera comparé avec les descriptions abstraites de services lors de la découverte de services (cf. figure 3.5, partie (b)) afin d'identifier les services candidats possibles (cf. figure 3.5, partie (c)). Vient enfin la contractualisation du service (cf. figure 3.5, partie 2), qui consiste en la description précise des capacités des services découverts dans la première phase (cf.figure 3.5, partie (d)), et la précision d'interactions entre le fournisseur et le demandeur de service en mentionnant tous les inputs, les préconditions et les postconditions).

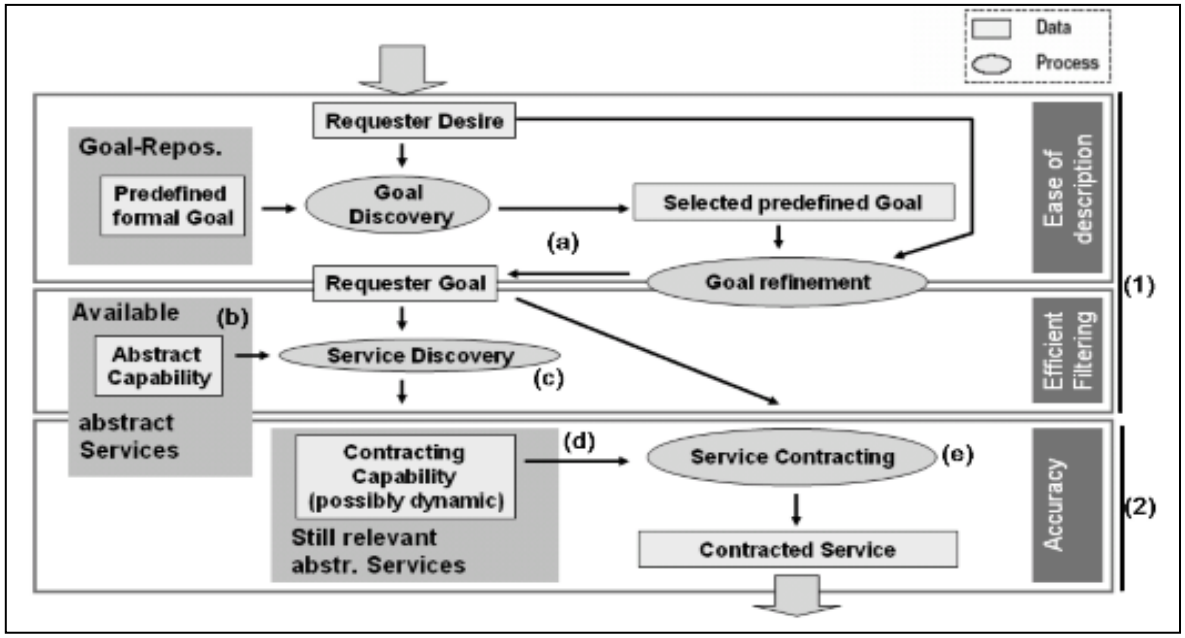


Figure 3.5: Modèle de localisation automatique et sémantique de services. [Source : Keller et al., 2005].

L'approche proposée distingue cinq degrés d'appariement appelés degrés d'appariement intentionnel :

- `Match` si le service offert satisfait complètement la requête,
- `PossMatch` lorsque le service offert peut satisfaire complètement la requête. A cause d'une incomplétude dans la description disponible, la satisfaction complète de la requête ne peut être garantie. Ce degré d'appariement n'est conclu qu'à la phase de contractualisation,
- `ParMatch` si le service offert satisfait la requête, mais partiellement. Il offre certains aspects requis mais pas la totalité.
- `PossParMatch` lorsque le service offert peut satisfaire partiellement la requête. A cause d'une incomplétude dans la description disponible, ceci ne peut être garanti. Ce degré d'appariement est conclu à la phase de contractualisation,
- `NoMatch` si aucun des précédents cas n'est identifié, l'offre est complètement non pertinente par rapport à la requête.

Küster et König-Ries ont essayé d'étendre ce travail au niveau des degrés d'appariement et ils ont rajouté deux degrés : RelationMatch et ExcessMatch pour mesurer des pertinences graduées (Graded Relevances, GR) [Küster et König-Ries, 2008].

3.2.2.4 PSWSD

Vu et ses co-auteurs ont proposé en 2005 une approche de découverte de services Web sémantiques fondée sur le réseau P2P (P2P-based Semantic Web Service Discovery, PSWSD) [Vu et al., 2005]. L'approche utilise une architecture déductive pour la découverte de services Web dans un réseau P2P. Les descriptions des services Web sont fournies selon l'ontologie WSMO au moyen de techniques spécifiques [Sivashanmugam et al., 2003] et sont publiées dans divers registres repartis dans le réseau P2P. Un utilisateur cherchant un service Web, avec certaines contraintes de qualité de service (QoS), peut interroger n'importe quel registre du réseau. Celui-ci dirigera la requête reçue vers le(s) registre(s) pouvant la satisfaire. Les fonctionnalités du service demandé sont ensuite extraites de la requête et envoyées au module matchmaker. Le matchmaker sélectionne les descriptions des services qui s'apparient sémantiquement avec la requête de l'utilisateur et transmet les résultats à l'utilisateur qui choisit le service à invoquer.

Dans ce système, les utilisateurs sont amenés à faire leurs évaluations des services Web en termes de QoS et à envoyer leurs feedbacks au registre (*Registry peer*) contenant les descriptions de ces services pour des utilisations ultérieures. Des agents de surveillance de qualités de service sont utilisés dans le but de vérifier l'authenticité des évaluations des utilisateurs.

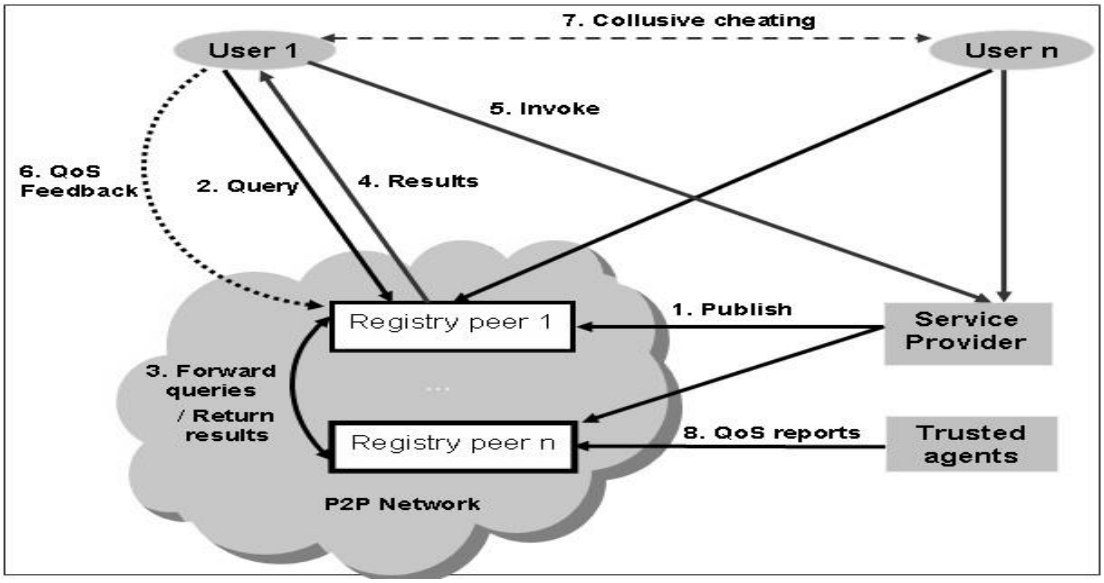


Figure 3.6 : Découverte sémantique de services Web fondée sur un réseau P2P. [Source : Vu et al., 2005]

3.2.2.5 CASD

Dans un autre travail de découverte de services tenant compte du contexte (Context Aware Service Discovery, CASD) [Doulkeridis et al., 2005], le module de découverte utilise des ontologies du domaine pour déterminer les catégories des services qui ont une relation sémantique avec la requête de l'utilisateur. Lorsque l'utilisateur formule sa requête (Q_{usr}) en termes de mots-clés, une autre requête plus riche (Q_{ctx}) est générée et sera attachée à cette dernière. La requête Q_{usr}permet de trouver les services Web qui ont une relation sémantique avec les termes de recherche de l'utilisateur, tandis que la requête Q_{ctx} joue le rôle de filtre permettant de sélectionner les services qui correspondent au contexte de l'utilisateur. Les requêtes formulées par les utilisateurs sont exprimées en SPARQL [SPARQL, 2008] qui est un langage de requête compatible avec les graphes OWL. Une version distribuée de cette approche a été proposée par Arabshian et Schulzrinne [Arabsdhian and Schulzrinne, 2004] qui ont présenté une architecture agent distribuée et tenant compte du contexte pour la découverte de services (Distributed Context Aware Agent architecture for Service Discovery, DCAASD).

3.2.3. Approche hybride

L'approche hybride utilise des mécanismes déductifs intégrant les méthodes de calcul de distances. L'idée est de remédier aux limites de chacun de ces deux mécanismes en les combinant. Plusieurs travaux [Rey, 2002] [Kiefer et al., 2007] [Kiefer et Bernstein, 2008] [Bertoli, 2004] [Kaufer et Klusch, 2006] [Klusch et Kapahnke, 2008] optent pour cette approche. Nous notons que d'autres travaux fondés sur le RàPC s'inscrivent également dans la catégorie d'approches hybrides. Nous en présentons une étude détaillée dans la section 3.2.4 vu qu'elles cadrent étroitement avec notre approche.

3.2.3.1 Approche basée sur la logique de description

Rey propose une approche hybride de découverte sémantique des services Web, fondée sur la Logique de Description (LD) [Rey, 2002]. Il définit le problème de découverte de services Web comme suit : « ayant une requête et des services exprimés dans une logique de description, comment trouver les combinaisons des services qui couvrent au mieux la requête, tels que le maximum d'informations de la requête se retrouve dans les combinaisons des services et que le minimum d'informations en plus soit apporté par ces combinaisons ».
Le processus de découverte consiste en une recherche des meilleures couvertures d'un concept de description en utilisant une certaine terminologie. Il cherche pour une description Q et une terminologie T, une description E qui couvre le plus la description Q en utilisant T [Rey et al., 2002].

Ainsi, pour réécrire la requête, l'auteur utilise les travaux de Teege [Teege, 1994] qui propose une classe de langages pour lesquels la différence entre concepts peut être calculée de la même manière qu'une différence ensembliste grâce à des descriptions sous forme clausale. Il choisit un langage de représentation de cette classe pour modéliser les services Web et leur appliquer ce raisonnement, même si son expressivité est réduite. Pour rechercher les meilleures couvertures d'un concept en utilisant une terminologie, l'auteur utilise un algorithme issu de la théorie des hypergraphes. L'algorithme utilisé effectue une recherche équivalente à la recherche des transversaux minimaux de coût minimal dans un hypergraphe où les sommets sont les services Web et les arêtes sont les clauses de la requête.

3.2.3.2 OWLS-MX

OWLS-MX [Klusch et al., 2006] est un matchmaker sémantique hybride qui réalise un IO-matching entre des profils de services OWL-S.

En plus de l'appariement classique dont les résultats (Exact ou Fail) indiquent que les éléments appariés sont exactement les mêmes ou n'ont par contre aucun lien, ce matchmaker propose cinq autres catégories d'appariement déductifs (Plug-In, Subsumes et Subsumed-By) et hybrides (Logic-based Fail et Nearest-neighbor). Ces dernières sont fondées sur la logique et l'utilisation de la similarité syntaxique. En réponse à une requête, l'algorithme de matching utilisé retourne un ensemble ordonné de services découverts à qui sont associés un degré d'appariement et une valeur de similarité syntaxique avec la requête.

$LSC(C)$ 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 3.7 : Les degrés d'appariement dans OWLS-MX. [Source : Klusch et al., 2006].

L'appariement sémantique (fondé sur la logique) des inputs et des outputs des services est notamment appuyé par le calcul de leur similarité syntaxique aussi.
Dans la littérature, différentes variantes de l'algorithme de matching utilisé par OWLS-MX (OWLS-M1, OWLS-M2, OWLS-M3 et OWLS-M4) ont été implémentées en optant pour des mesures différentes de similarité syntaxique [Klusch et al., 2006].

3.2.3.3 OWLS-iMatcher2

OWLS-iMatcher2 [Kiefer et al., 2007] [Kiefer, 2009] [Kiefer et Bernstein, 2008] est une approche hybride de découverte qui opte à la fois pour un appariement déductif des concepts d'inputs/outputs et un appariement algébrique utilisé pour le calcul de la similarité textuelle

des noms et des signatures des services [Bernstein et al., 2005] [Bernstein et Kiefer., 2005]. L'agrégation de ces deux appariements pour une évaluation globale utilise une valeur binaire de pertinence sémantique ainsi que différentes valeurs de métriques de similarité telles que : Bi-Gram [Kiefer, 2009], les mesures de Levenshtein [Levenshtein, 1996] [Gusfield, 1997], Monge-Elkan [Kiefer, 2009] et Jaro [Winkler et Thibaudeau, 1991] [Kiefer, 2009]. L'algorithme d'appariement utilisé calcule les valeurs de similarité syntaxique entre une requête donnée et tous les services disponibles, puis utilise un modèle mathématique de régression pour prédire la valeur d'agrégation d'appariements pour chaque service. Les services découverts sont renvoyés à l'utilisateur dans l'ordre décroissant des valeurs d'appariement obtenues. Une évaluation statistique des résultats retournés aussi bien que leur présentation graphique lui permettent de les interpréter et les évaluer facilement.

3.2.3.4 FC-Match

Dans l'approche hybride FC-Match (Functional Comparison) [Bertoli, 2004], les concepts du service et de la requête à apparier sont présentés en OWL-DL (Web Ontology Language Description Logics). Les concepts utilisés pour décrire un service apportent des informations sur les éléments : Category, Operation, Input et Output.

Un service est perçu 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 S un service, C1 le concept associé au paramètre catégorie du profil de S, C2 le concept associé au paramètre opération, C3 le concept associé au paramètre input et 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)
```

Cette approche opte également pour un appariement sémantique déductif et un appariement algébrique fondé sur le calcul de similarité syntaxique. Le calcul de la valeur globale d'appariement est effectué en agrégeant le résultat de l'appariement fondé sur la subsomption logique des concepts entre le service et la requête, et la valeur du coefficient de similarité syntaxique.

3.2.3.5 WSMO-MX

L'approche WSMO-MX [Kaufer et Klusch, 2006], quant à elle, est une approche hybride d'appariement de services décrits en WSML-MX [Klusch et al., 2008]. Elle opte pour un appariement sémantique [Klusch et al., 2006] et syntaxique et procède par des techniques d'appariement de graphes orienté objet héritées du DSD-Matchmaker [Klein et König-ries, 2004 a] et d'appariement intentionnel de services [Keller et al., 2005].

Le degré global d'appariement est calculé par l'agrégation de quatre valeurs représentant les valeurs d'appariement des types ontologiques, d'appariement logique des contraintes spécifiées en F-logic (Frame Logic), d'appariement des noms de relations et de mesure de similarité syntaxique.

Par ailleurs, l'appariement sémantique effectué est catégorique. Il porte sur les préconditions et les post-conditions. Le tableau 3.1 illustre les degrés d'appariement, leur ordre, leur symbole et leur signification par rapport aux états de précondition et postcondition.
Soient **G** un goal (un service requis) et **W** un service WSMO (un service offert). Le tableau 3.1 donne la liste des degrés d'appariement utilisés dans l'approche WSMO-MX, dans l'ordre décroissant de leurs valeurs d'appariement, allant de l'équivalence à l'échec. Les degrés `plugin` et `inverse-plugin` sont définis différemment pour les préconditions et postconditions.

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)	$\mathcal{G} \cap \mathcal{W} = \emptyset$	

Tableau 3.1: Les degrés d'appariement dans WSMO-MX [Source : Kaufer et Klush, 2006].

3.2.3.6 SAWSDL-MX

SAWSDL-MX [Klusch et Kapahnke, 2008] est un matchmaker inspiré des matchmakers OWLS-MX [Klusch et al., 2006] et WSMO-MX [Kaufer et Klusch, 2006] qui permet de découvrir des services décrits en SAWSDL [SAWSDL, 2011]. Il opte à la fois pour un appariement logique fondé sur le raisonnement par subsomption et un appariement syntaxique fondé sur des techniques de recherche d'information.

Partant du fait qu'une description SAWSDL [Kuster et Konig-Ries, 2007] est une extension d'une description WSDL, l'appariement adopté couvre les éléments de description suivants :

- `interface` pour la découverte des services spécifiés en WSDL 2.0 et `portType` pour les services spécifiés en WSDL 1.1,
- `operation`, `input`, `output` pour la découverte des services WSDL 1.1 ou WSDL 2.0 [Farell et Lausen, 2007b].

Pour réaliser l'appariement des interfaces, le matchmaker effectue des appariements sur des graphes bipartis, et pour apparier des opérations, il utilise différentes techniques syntaxiques [IDEAL, 2011] [Fuglede et Topsoe, 2004], issues de la logique (en exploitant les annotations sémantiques figurant dans les fichiers SAWSDL de description des services) aussi bien qu'hybrides.

Le processus d'appariement classe les résultats obtenus, selon leurs degrés d'appariement, en différentes catégories qui se répartissent en deux groupes [Klusch et Kapahnke, 2008].
Le groupe de catégories Exact, Plug-in, Subsumes et Fail est utilisé pour classer les résultats suite à un appariement sémantique, tandis que le groupe de catégories Subsumed-by et Nearest-neighbour est utilisé pour classer ceux d'autres résultats pour lesquels l'appariement sémantique a échoué. Ces catégories sont en fait obtenues en appliquant un appariement syntaxique. La catégorie Subsumed-by est obtenue suite uniquement aux résultats de calcul de similarité syntaxique. La catégorie Nearest-neighbour quant à elle, est obtenue suite à l'agrégation des résultats des appariements sémantique et syntaxique. Elle permet de compenser les résultats faux positifs dans l'appariement effectué.

3.2.4. Etude comparative

Le recours à une troisième approche de raisonnement hybride permet de déduire que les approches algébriques et déductives peuvent être combinées pour combler leurs limites respectives et consolider leurs avantages. En analysant ces trois classes d'approches, nous constatons qu'effectivement l'approche déductive applique des règles prédéfinies et exploite la logique pour raisonner en termes de liens sémantiques et pour déduire de façon plus précise la similitude ou non d'une paire de concepts. Par contre, l'approche algébrique calcule généralement cette similitude en se contentant du calcul du chemin (le nombre d'arcs et parfois le poids de ces arcs) entre ces concepts (représentés comme nœuds de graphe). Nous

pensons que se fier au nombre d'arcs qui lient un concept à un autre dans un graphe n'est toujours pas un moyen adéquat pour déterminer avec précision leur degré de similitude et constitue un aspect que l'on peut reprocher à l'approche algébrique. Toutefois, à un autre niveau, nous constatons que pour le calcul de la similitude globale des services requis et fournis, cette approche utilise des formules algébriques adaptées au calcul de distances obtenues par l'agrégation de distances partielles. Ces formules permettent d'obtenir des mesures dont la qualité est prouvée algébriquement. Chacune des deux approches algébrique et déductive a alors son propre avantage qui lorsqu'il est combiné à l'autre dans une approche hybride, permet de maximiser la précision du calcul de similitude entre les services requis et fournis.

La classification des travaux existants dans la littérature selon un formalisme, et par la suite, le raisonnement adopté pour effectuer l'appariement est certes une brique importante dans le processus de leur évaluation. Elle renseigne sur leur efficience qui dépend essentiellement de la technique d'appariement qu'elles adoptent. Cependant, elle ne s'avère pas suffisante pour évaluer leur adaptation au contexte des services Web. Pour ce faire, d'autres critères doivent être pris en considération.

Aussi, identifions-nous cinq critères pour comparer les divers travaux présentés auparavant. Ils sont relatifs essentiellement aux mécanismes d'appariement, de formalisation et de description des services Web objets de la découverte. Ces mécanismes constituent notamment les piliers de toute approche de découverte de services Web. Aussi, les critères de comparaison identifiés sont-ils les suivants : le type d'appariement, l'objet d'appariement, le niveau d'appariement, le formalisme et le type de services.

- **Le type d'appariement**

Le type d'appariement renseigne sur sa profondeur. Notamment, les concepts utilisés pour décrire les propriétés d'un service Web et ceux utilisés pour exprimer des propriétés requises dans une requête ne sont pas nécessairement les mêmes, autrement, ils ne sont pas forcément syntaxiquement similaires. Aussi, un appariement profond ou encore « intelligent » ne devrait pas se limiter au niveau syntaxique. Il devrait par contre couvrir l'aspect sémantique ou encore couvrir les deux niveaux si cela est nécessaire.

- **L'objet d'appariement**

L'objet d'appariement renseigne sur les éléments appariés parmi ceux figurant dans les descriptions du service Web requis et celui disponible. Cette information est importante puisqu'elle permet de savoir sur quelle base l'appariement est fondé. Notamment, la qualité des résultats d'appariement peut être évaluée en fonction des éléments qui ont fait l'objet de cet appariement. Un service Web dont les entrées et sorties correspondent bien aux entrées et sorties du service Web requis n'est pas nécessairement le service adéquat pour le client s'il paraît qu'il exige des préconditions d'exécution que ce dernier ne peut pas garantir. En outre, même si un appariement global de l'ensemble de propriétés fonctionnelles désigne un service Web comme service approprié pour répondre à une requête, celui-ci peut ne pas l'être vraiment si la requête comprend également des contraintes non-fonctionnelles exigées par le client.

- **Le niveau d'appariement**

Le niveau d'appariement informe sur le degré de similitude cherchée lors du processus d'appariement. Certains travaux considèrent qu'un élément ne s'apparie avec son correspondant que lorsque leur similitude est forte (exacte). D'autres proposent plusieurs catégories de similitudes (Plug-in, subsume, etc.). L'élément apparié n'a pas nécessairement à être le même que son correspondant mais peut, par exemple, le couvrir ou en être une instance. Nous pensons que cette flexibilité d'appariement permet, en cas d'absence de services Web qui répondent exactement à une requête, de découvrir d'autres qui peuvent couvrir partiellement les besoins exprimés ou les encapsuler.

- **Le critère de formalisme**

Le critère de formalisme renseigne sur le formalisme utilisé pour représenter les services Web et les requêtes. Dans la pratique, la manière dont un problème est formalisé détermine souvent le type de raisonnement à adopter pour le résoudre. Or, le raisonnement pourrait avoir un impact direct sur la performance et son déploiement en pratique, et par la suite sur la qualité des résultats qui en dépendent. Aussi, la performance et le déploiement du processus de découverte des services Web, aussi bien que la pertinence des services Web découverts en réponse à une requête, pourraient-ils dépendre du type de raisonnement utilisé, et par récurrence, du formalisme adopté.

- **Le type de services**

Le type de services désigne l'ontologie ou le langage utilisé pour décrire les services Web. Le formalisme adopté représente ces services conformément à l'ontologie ou au langage utilisé. Pour pouvoir réaliser des appariements sémantiques, la plupart de travaux existants recourent à l'utilisation des ontologies OWL-S ou WSMO comme modèles sémantiques pour décrire les services Web. Cependant, il en résulte que l'approche proposée est exclusivement dépendante de l'ontologie choisie et donc spécifique pour la découverte d'un type précis de services Web (ceux décrits conformément à cette ontologie). Ces ontologies sont le produit de travaux menés par des laboratoires de recherche ayant exploité les résultats du Web sémantique dans le but de couvrir l'aspect sémantique des services Web. Cependant, ces composants logiciels sont en réalité publiés avec des descriptions conformes aux standards W3C dont l'ancêtre est le langage WSDL. Dans un but de cibler directement des services disponibles sur le Web, nous pensons que les formalismes de représentation des services Web doivent respecter les standards de description définis par W3C. De plus, leur traitement sémantique doit être indépendant de tout type particulier d'ontologies.

Le choix de l'ensemble des critères présentés ci-dessus n'est pas aléatoire. Nous les avons identifiés en fonction de cinq métriques considérées comme principaux indicateurs de performance de tout processus, à savoir, l'efficacité, l'efficience, la flexibilité et l'alignement avec les standards. Notamment, les critères type et objet d'appariement, tels qu'ils étaient définis, permettent de mesurer l'efficacité du processus de découverte qui dépend étroitement de l'efficacité du processus d'appariement. Le critère de formalisme est un indicateur de son efficience puisque le raisonnement adopté en dépend. Le niveau d'appariement, quant à lui est un critère qui permet de mesurer la flexibilité du processus de découverte. Finalement, le type de services ciblés est un indicateur qui permet de dégager les travaux qui s'alignent avec les standards. Notamment, dans le cas des approches pour les services Web, l'alignement avec les standards garantit à ces approches l'indépendance des modèles de description particuliers (ontologies ou autres).

Le tableau 3.2 présente une synthèse des résultats de l'étude comparative des travaux que nous avons présentés auparavant dans ce chapitre. Ce tableau comparatif permet en fait de déterminer jusqu'à quel point chacun de ces travaux serait en mesure d'être qualifié d'efficace, efficient, flexible et aligné avec les normes, et ce, en évaluant ses caractéristiques.

		Formalisme	Type d'appariement	Objet d'appariement	Niveau d'appariement	Type de services
APPROCHE ALGEBRIQUE	**iMatcher1**	RDF Graphe	Syntaxique	Propriétés fonctionnelles (Profil du service)	Exact, Fail	OWL_S
	AASDU	Ensemble de termes	Syntaxique	Propriétés fonctionnelles (Inputs/Outputs)	Exact, Fail	WSDL
	DSD-Matchmaker	Graphes DSD	Syntaxique	Propriétés fonctionnelles (Inputs/Outputs)	Exact, Fail	Diane Service Desciption
APPROCHE DEDUCTIVE	**WSC**	Concepts DAML	Sémantique	Propriétés fonctionnelles (Inputs/Outputs)	Exact, Plug-in, Subsume, Fail	DAML
	OWLS-M	OWL-S	Sémantique	Propriétés fonctionnelles (Inputs/Outputs)	Equivalent, Subsume, Unknown, Fail	OWL-S
	ALS	Logique (Goals + Axiomes)	Sémantique	Propriétés fonctionnelles (Inputs/Outputs/Goals)	Match, PossMatch, ParMatch, PossParMatch, No-M., Excess M.	WSMO
	PSWSD	WSMO	Sémantique	Propriétés fonctionnelles (Goals, Préconditions/ Postconditions)	Exact, Fail	WSMO
	CASD	Graphe OWL	Sémantique	Propriétés fonctionnelles (Inputs/Outputs)	Exact, Fail	CASD
APPROCHE HYBRIDE	**LD**	Description logique + Hypergraphe	Sémantique	Propriétés fonctionnelles (Inputs/Outputs)	Exact, Fail	Non spécifié
	OWLS-MX	OWL-S	Sémantique	Propriétés fonctionnelles (Inputs/Outputs)	Equivalent, Subsume, Subsumed by, Logic-Based fail, N.N., Fail	OWL-S
	OWLS-iMatcher2	OWL-S	Sémantique et syntaxique	Propriétés fonctionnelles (Inputs/Outputs)	Exact	OWL-S
	FC-Match	OWL-S + OWL-DL	Sémantique et syntaxique	Propriétés fonctionnelles (Inputs/Outputs/ Categories/Operations)	Exact	OWL-S
	WSMO-MX	WSMO	Sémantique et syntaxique	Propriétés fonctionnelles (Service Goals, Préconditions/ Postconditions)	Equivalent, Plug-in, Inverse Plug-in, Intersection, Fuzzy similarity, Neutral, Disjunction	WSMO
	SAWSDL-MX	SAWSDL	Sémantique et syntaxique	Propriétés fonctionnelles (Inputs/Outputs/Operations)	Equivalent, Subsume, Subsumed by, N.N., Fail	SAWSDL

Tableau 3.2 : Tableau comparatif des approches de découverte de services Web

Synthèse :

Le succès des services Web publiés dépend des processus de découverte. Le présent travail a été mené dans le souci de dégager les caractéristiques qu'une approche de découverte de services Web doit posséder pour être qualifiée d'efficace, efficiente, flexible et alignée avec les standards. Dans le but d'identifier ces caractéristiques, nous avons veillé à mettre en relief les mécanismes, techniques, modèles ou langages adoptés par les travaux réalisés dans ce domaine. Cela a permis de découvrir les moyens explorés, de les comparer et de dégager ceux devant être adoptés par une approche de découverte pouvant être qualifiée d'adéquate.

A travers l'étude menée, nous avons constaté d'abord que les travaux réalisés dans la littérature optent pour trois approches en termes de formalisme et raisonnement adoptés pour représenter une requête de service Web et effectuer l'appariement : l'approche algébrique, l'approche déductive et l'approche hybride. Les travaux de l'approche hybride cherchent à combler les limites des travaux des deux approches algébrique et déductive en combinant leurs techniques et mécanismes.

Par ailleurs, comme nous le constatons à partir du tableau comparatif des travaux de découverte de services Web que nous avons présentés, le formalisme et raisonnement adopté pour l'appariement des services Web forment un critère crucial et primaire qui n'est tout de même pas exclusif pour l'évaluation des approches de découverte des services Web. D'autres critères, à savoir le niveau, l'objet et le type d'appariement aussi bien que le type de services ciblés, s'avèrent également importants pour évaluer l'adaptation de ces approches au contexte des services Web. En analysant ces critères renseignés dans le tableau 3.2, nous pouvons conclure qu'une approche adaptée à la découverte de services Web doit :

- Cibler de préférence des services Web standards en s'alignant avec les standards W3C.
- Intégrer les propriétés non-fonctionnelles aux propriétés fonctionnelles des services Web comme objets d'appariement, afin de répondre au mieux aux exigences de l'utilisateur.
- Opter pour un appariement plutôt sémantique pour ne pas écarter d'éventuels résultats comme dans le cas de l'appariement syntaxique.
- Opter pour un type d'appariement assurant un minimum de flexibilité. En d'autres termes, ne pas se limiter à une approche exacte dans le but de permettre la découverte d'éventuels services pouvant répondre à la requête même si leurs propriétés ne correspondent pas exactement à celles exigées par la requête.

Parmi les propositions adoptant l'approche hybride, nous relevons l'émergence depuis 2006 du recours au paradigme de Raisonnement à Partir des Cas (RàPC). Rappelons que le RàPC ou Case Based Reasoning (CBR) s'appuie sur la réutilisation de l'expérience formée lors de la résolution d'une situation problème pour la résolution d'une situation problème similaire.

L'idée est que la découverte de services Web, en particulier l'appariement automatique des services Web peut encore être amélioré en tenant compte des expériences de recherche et d'appariement déjà effectuées. Par conséquent, ces approches exploitent une représentation des connaissances du domaine de services Web pour capturer les expériences de découverte des services et les utiliser dans un nouveau processus d'appariement. La prise en compte des expériences passées permet de réduire le temps de réponse du processus d'appariement qui se limite aux requêtes déjà effectuées.

Dans la section suivante, nous détaillons ces approches fondées sur le RàPC étant donné que c'est une orientation des recherches qui nous semble prometteuse et que nous avons adoptée.

3.3. Travaux de découverte de services Web à base de RàPC

Très peu de travaux dans la littérature se sont intéressés à l'optimisation du temps de découverte ou composition de services Web en exploitant le raisonnement à Partir des Cas RàPC [Osman et al., 2006 a] [Osman et al., 2006 b] [Osman et al., 2009][Lajmi et al. , 2006 a] [Lajmi et al., 2006 b][Lajmi et al., 2007][Lajmi et al., 2011][Wang et Cao, 2007] [De Franco Rosa et De Oliveira , 2008]. Nous étudions leurs détails dans les sections qui suivent dans le souci de dégager leurs caractéristiques et pouvoir les comparer à notre travail. Signalons que les travaux postérieurs à 2011 se concentrent à notre connaissance sur la composition de services Web.

3.3.1. S-CBR

Osman, Thakker et Al-Dabass [Osman et al., 2006 a] [Osman et al., 2006 b] [Osman et al., 2009] présentent leur système S-CBR dans lequel les expériences d'exécution des services Web sont modélisées comme des cas qui représentent les propriétés fonctionnelles et non-fonctionnelles des services Web en utilisant une description sémantique OWL. Dans ce travail, la base de cas sert au stockage des expériences. Elle est identique au registre de services Web qui stocke les références de ces services, mais elle décrit en plus le comportement d'exécution.

Dans le système S-CBR, deux rôles principaux sont distingués : l'administrateur de cas qui est responsable de la maintenance de la base de cas par l'ajout ou la suppression de cas dans la base, et l'utilisateur, qui cherche dans la base de cas la solution d'un problème. Ce dernier cherche en fait à découvrir un service Web qui satisfait sa requête. La figure 3.8 décrit l'architecture de ce système.

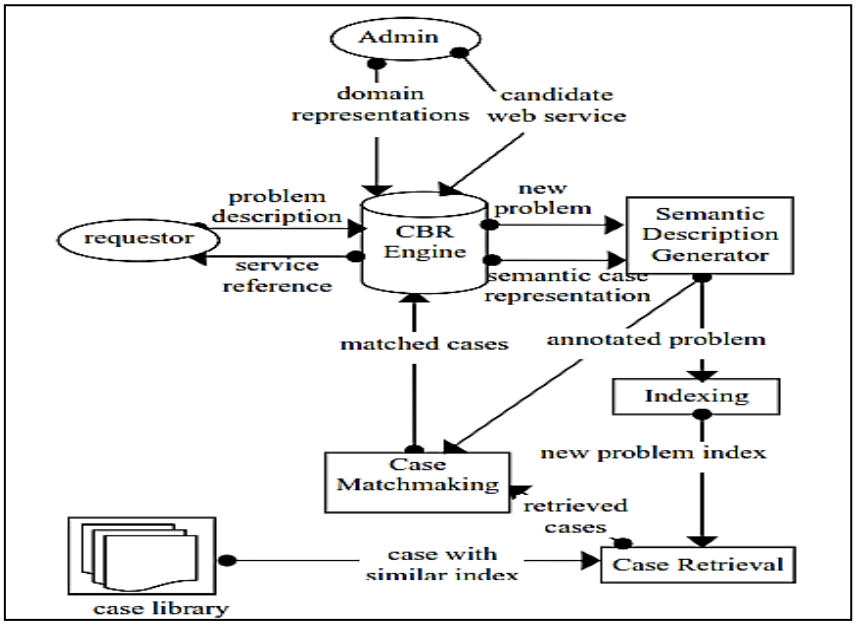


Figure 3.8 : Architecture de S-CBR. Source [Osman et al., 2006 b]

Initialement, l'administrateur remplit le référentiel avec des cas sémantiques et des formats de représentation de cas relatifs à différents domaines d'application. Par la suite, les requêtes des utilisateurs seront aussi annotées sémantiquement selon un même format que le cas du référentiel et du domaine auquel elles réfèrent. En effet, après avoir reçu la description d'un problème, S-CBR commence la recherche pour trouver des services adaptés qui correspondent à la demande. Il passe ainsi la description du problème et le format sémantique de représentation de cas considérés, au module générateur de description sémantique. Celui-ci annote le nouveau problème selon le format de représentation approprié.

Le vocabulaire utilisé pour représenter les propriétés d'un service Web est spécifique à un domaine particulier et devrait couvrir les connaissances associées avec le contexte de ce domaine. Les propriétés fonctionnelles représentent les paramètres d'entrée et de sortie du service Web et les propriétés non-fonctionnelles représentent des contraintes ou des préférences pour certains paramètres de ce service (par exemple la monnaie et la qualité de service attendue).

Dans ce travail, les auteurs proposent de formaliser leurs cas par des structures de frame (*frame-structures*) où un *frame* comprend des *slots* et des *fillers*. Les *slots* sont constitués de plusieurs dimensions qui représentent les attributs dont les valeurs sont issues du vocabulaire prédéfini, tandis que les *fillers* sont la plage de valeurs possibles pour ces attributs. Ces structures ne sont pas généralisées et dépendent fortement du domaine d'application.

Pour illustrer l'utilisation du frame dans le domaine de gestion de vols « Travel domain », nous donnons l'exemple du tableau 3.3. Dans ce tableau, les slots sont définis par Travel Request, Preferences et Features. Quant aux dimensions, elles sont représentées par City Departure, City Arrival, On Price range, On Currency et Travel Regions. Enfin, New York, Boston, 220, USD, Domestic représentent les fillers.

Travel Request	
City Departure	New York ([City (USA [Country])]) $_{\text{is-city-of}}$
City Arrival	Boston ([City (USA [Country])]) $_{\text{is-city-of}}$
Preferences	
On Price range	220
On Currency	USD ([Currency]) $_{\text{code}}$
Features	
Travel Regions	Domestic ([Travel Regions])
Class = [class], Instance = instance ([class]), Property = $_{\text{properties}}$	

Tableau 3.3: Exemple de cas représenté par frame dans le domaine de gestion de vols.

Afin de permettre la récupération rapide des cas appropriés lors de la procédure de recherche, les auteurs proposent d'organiser la base de cas en fonction de certains termes du vocabulaire représentant des caractéristiques (*features*) des services, ce qui permet de la subdiviser en un nombre de partitions égale au nombre d'instances du vocabulaire des *features* utilisé [Osman et al., 2006a]. Ainsi, sur la base du vocabulaire utilisé dans la description du nouveau problème, ce système détermine la partition de la base de cas qui concerne ce problème. Ensuite, un algorithme de matching est appliqué sur les cas de la partition sélectionnée. Pour ce faire, ils appliquent la fonction de similarité suivante qui permet de calculer le degré global de similarité pour chaque cas remémoré.

$$\frac{\sum_{i=1}^{n} W_i \ * sim\ (f_i^N, f_i^R)}{\sum_{i=1}^{n} W_i}$$

Où n est le nombre de dimensions, $W_i$est l'importance de la dimension i, sim est la fonction de similarité pour les primitives, et f_i^N et f_i^R sont respectivement les valeurs du vocabulaire correspondant à l'attribut f_i dans le nouveau problème et dans le cas.

3.3.2. WeSCo_CBR

Dans le but de réaliser une plateforme pour la composition de services web, Lajmi et ses co-auteurs [Lajmi et al., 2006 a] [Lajmi et al., 2006 b] [Lajmi et al., 2007] ont proposé le système WeSCo_CBR basé principalement sur les ontologies et le RàPC.

Pour apporter un guidage semi-automatique à l'utilisateur, les auteurs ont défini une ontologie OWL-S qui décrit les différentes fonctionnalités des services web à base de OWL. Ainsi, dans le but de faciliter le traitement d'une requête utilisateur, ils procèdent par la transformation de cette requête sous une forme compréhensible et manipulable par la machine. Cette étape permet alors de représenter la requête par une formule ontologique sous la forme d'un ensemble de concepts de l'ontologie que les auteurs ont défini. Pour ce faire, ils divisent la requête en trois parties :

- Instances : Une requête peut contenir un ensemble de données. Ces données peuvent être considérées comme des valeurs d'attributs d'un ou plusieurs objets. La partie des instances de la requête est représentée conformément aux classes de ces objets.
- Variables : Une requête peut contenir des variables. Ces variables peuvent être considérées comme des attributs d'une ou plusieurs classes qui représentent la partie des variables de la requête.
- Activités : C'est l'ensemble des activités abstraites (fonctionnalités) d'une requête. En effet, les activités d'une demande sont déduites des classes de variables et d'instance.

Après avoir défini les instances et les variables de la requête, la recherche d'activités est lancée afin d'obtenir toutes les activités requises.

Dans le système WeSCo_CBR, un cas est composé des trois éléments suivants [Lajmi et al., 2011]:

- Le problème qui regroupe quatre parties, à savoir le profil utilisateur, les activités, les variables et les instances ;
- La solution qui est l'ensemble des activités sous forme d'un schéma de composition;
- L'évaluation qui est le taux de pertinence de la solution.

Nous notons que les auteurs ne fournissent pas les détails nécessaires ou des exemples permettant de mieux éclaircir la manière dont ils formalisent une requête et un cas.

Le processus de réutilisation dans WeSCo-CBR consiste, pour une nouvelle requête, à récupérer un cas antérieur similaire mémorisé et éventuellement évaluer et mémoriser le nouveau cas.

Une requête donnée peut exiger un ensemble d'activités abstraites reflétant des services Web de différents domaines métiers. C'est ainsi que, les auteurs proposent d'organiser la base de cas par domaine métier afin de réduire l'espace de recherche, pour une activité abstraite donnée, au domaine métier correspondant.

Pour chaque nouvelle requête, le système cherche les cas similaires et en sélectionne le cas pertinent, en fonction de sa ressemblance avec la requête. Pour ce faire, le système utilise la fonction de similarité suivante :

$$Sim_v(NC, MC) = \frac{\sum_{i=1}^{NV} Sim_v(NCV_i, MC)}{\max(NV, MV)}$$

Où :

- NC est un nouveau cas cible (requête),
- MC est un cas source,
- NV and MV sont, respectivement, le nombre de variables de NC et de MC.
- NCV_i est l'une des variables du nouveau cas cible (requête)
- Max est la fonction qui retourne le maximum entre deux valeurs comparées.
- Sim est la fonction de similarité entre deux variables.

3.3.3. CBR/OWL-S Matching Engine

Dans le but d'améliorer la découverte des services Web, Wang et Cao ont introduit un système à base de RàPC nommé CBR / OWL-S Matching Engine [Wang et Cao, 2007]. Pour mieux cibler les services Web cherchés, le moteur d'appariement CBR/OWL-S utilise les ontologies pour une recherche sur le Web sémantique fondée sur le calcul de distance pour le calcul de similarité.

L'architecture du moteur d'appariement CBR/OWL-S Matching Engine est représentée dans la figure 3.9. Le cœur du moteur d'appariement CBR/OWL-S est le moteur de raisonnement RàPC avec lequel est intégré le raisonneur OWL-S. En recevant une requête, le moteur d'appariement sélectionne les cas correspondant à cette requête à partir de la base de cas : le moteur de raisonnement RàPC se charge de calculer le degré de correspondance.

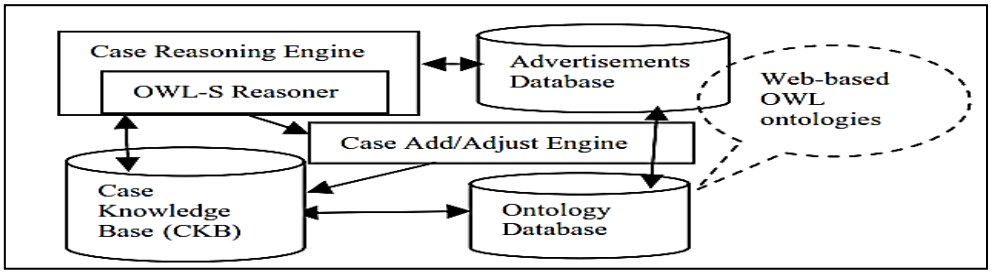


Figure 3.9: CBR / OWL-S Matching engine. Source [Wang et Cao, 2007]

Le cas sélectionné est renvoyé au demandeur. En cas d'échec d'appariement, le moteur d'appariement démarre la recherche d'un nouveau cas de services Web pouvant répondre à la requête traitée. Cela consiste à:

- Sélectionner des cas à partir de la base de publicités,
- Appeler le raisonneur OWL-S pour calculer le degré d'appariement en utilisant des ontologies.
- Sélectionner et envoyer au client le service dont le degré d'appariement est le plus élevé.

Pour chaque cas *f*, les auteurs définissent n dimensions pouvant décrire avec précision ses propriétés. Pour chaque dimension, ils attribuent un poids W reflétant son niveau d'importance dans le cas.

Le moteur de raisonnement de cas calcule le degré de similarité du cas cible par rapport aux cas sources de la base. L'algorithme du moteur de raisonnement adopte la formule de similarité suivante:

$$\text{similar}(f^I, f^R) = \frac{\left(\sum_{i=1}^{n} W_i \times sim\ (f_i^I, f_i^R)\right)}{\sum_{i=1}^{m} W_i}$$

Où :

- f^I : le cas cible,
- f^R : le cas remémoré,
- W_i : le poids de la ième dimension des cas f^I et f^R,
- f_i^I : une dimension du cas cible,
- f_i^R : une dimension du cas source,
- Sim (f_i^I, f_i^R) : la similarité entre les cas f^I et f^R au niveau de la dimension
- m le nombre total de dimensions ayant Sim $(f_i^I, f_i^R) \neq 0$
- et n le nombre total de dimensions.

3.3.4. Travail proposé par De Franco Rosa et De Oliveira, 2008

De Franco Rosa et De Oliveira [De Franco Rosa et De Oliveira, 2008] ont présenté un travail, fondé sur le RàPC, proposant la description sémantique des services et la recherche de cas basées sur les profils des utilisateurs.

Dans ce travail, chaque recherche de services peut être stockée comme un cas de recherche dans une base de données. Cependant, l'utilisateur a la possibilité de choisir de le stocker dans une base de cas particulière, qui reflète son profil. Notamment, trois bases de données sont utilisées. La première (UDDI) stocke des informations sur les services annotés sémantiquement selon SAWSDL, par des concepts d'une ontologie OWL. La seconde contient les cas de recherche (cas enregistrés par l'utilisateur comme des solutions satisfaisantes pour ses requêtes). La troisième (profil) regroupe les préférences d'un utilisateur parmi les cas satisfaisants.

Le moteur du système recherche les services en se basant sur des termes syntaxiques ou des concepts sémantiques. La recherche est effectuée d'abord dans la base du profil, puis la base de cas. Elle est en fait réalisée de façon purement syntaxique à ce niveau. En effet, la recherche sémantique n'est utilisée qu'au niveau de la base des services (UDDI). C'est à ce stade que le système calcule la similitude du contenu sémantique décrit dans l'annotation des services, avec la requête. Les services identifiés dans le registre UDDI seront stockés dans la base de recherche ou dans le profil de l'utilisateur.

3.3.5. Etude comparative

Nous identifions six critères pour comparer les divers travaux à base de RàPC présentés auparavant. Nous mettons le point sur des critères spécifiques pour l'évaluation des approches fondées sur ce type de raisonnement. En plus des critères d'une approche de découverte de services Web appropriée que nous avons conclus dans la synthèse de la section 3.2.4, nous prenons en considération des critères relatifs essentiellement aux mécanismes de formalisation des cas de découverte de services Web et d'organisation de la base de cas. Ces mécanismes constituent notamment des piliers de toute approche de découverte de services Web à base de RàPC. Aussi, les critères de comparaison que nous avons identifiés sont-ils les suivants : la représentation du cas, les objets d'appariement, l'annotation sémantique, le type d'appariement des cas, le niveau d'appariement et l'organisation de la base de cas.

- **La représentation du cas**

Ce critère correspond à la formalisation du cas de découverte de service Web. Autrement dit, il fait référence aux données ou connaissances représentées dans un cas. Nous pensons que la formalisation d'un cas doit de préférence être alignée avec les standards des services Web.

- **L'objet d'appariement**

Ce critère reflète la capacité de prise en considération, dans la formalisation du cas, des différentes propriétés pouvant constituer la requête du client, à savoir les propriétés fonctionnelles et non-fonctionnelles. Plus les propriétés prises en compte par l'approche sont nombreuses, meilleure est la réponse aux besoins des utilisateurs en considérant ces différentes propriétés comme objet d'appariement.

- **L'annotation sémantique**

L'annotation sémantique désigne le langage utilisé pour décrire sémantiquement les services Web et les cas de découverte. L'étude présentée au chapitre 2 dresse différents langages de description sémantique des services web et note l'avantage du standard W3C SAWSDL. La question posée est : est-ce que les annotations utilisées sont conformes à ce standard ?

- **Le type d'appariement des cas**

La remémoration de cas de services Web est la phase fondamentale du processus de découverte à base de RàPC. Elle est fondée sur l'appariement des cas qui peut être un appariement syntaxique ou sémantique. La pertinence des résultats et du processus de découverte dépend de la pertinence de l'appariement adopté et des mesures de similarités utilisées.

- **Le niveau d'appariement des cas**

Le niveau d'appariement des cas informe sur le degré de similitude cherchée lors du processus d'appariement (Exact, subsumes, etc.).

- **L'organisation de la base de cas**

Les cas sont classés dans une mémoire appelée base de cas. Afin de faciliter la recherche du cas le plus approprié au problème posé, il faut organiser la base de cas. Le choix de la méthode d'organisation est important pour la remémoration rapide des cas. En effet,

l'organisation de la base doit permettre d'accéder plus rapidement aux cas adéquats en fonction des éléments du contexte du problème cible. Notamment, lors de la recherche des cas, c'est la partie problème qui est sollicitée.

Le tableau 3.4 présente une synthèse des résultats de notre étude comparative des travaux à base de RàPC que nous avons présentés. Ce tableau comparatif permet en fait de montrer les caractéristiques de chacun de ces travaux.

	Représentation du cas	**Objet d'appariement**	**Annotation sémantique**	**Type d'appariement**	**Organisation de la base de cas**	**Niveau d'appariement**
S-CBR	(*prop fonct, prop. non-fonct, solution*)	- Propriétés fonctionnelles (Inputs/Outputs/ constraints/feature) - Propriétés non-fonctionnelles (preferences)	OWL-S	- Sémantique (Mesure Block city pondérée)	Partitionnement par caractéristiques	- Exact - Fail
WeSCo-CBR	(*User profile, Activities, Variables, Instances, ensemble d'activités, évaluation*)	- Propriétés fonctionnelles (Inputs/Outputs)	OWL-S	- Sémantique (Mesure de Manhattan adaptée)	Partitionnement par domaine métier	- Exact - Fail
CBR-OWLS	(*prop.fonct, solution*)	- Propriétés fonctionnelles (Inputs/Outputs)	OWL-S	- Sémantique (Mesure pondérée spécifique)	Aucun partitionnement	- Exact - Subclass - Superclass - Overlapping - Fail
De Franco Rosa & De Oliviera	(*prop.fonct, solution*)	- Propriétés fonctionnelles (Inputs/Outputs)	SAWSDL	- Syntaxique/ Sémantique (Mesure non spécifiée)	Aucun partitionnement	- Exact - Fail

Tableau 3.4: Tableau comparatif des approches de découverte de services Web à base de RàPC

Synthèse :

Les travaux présentés dans cette section adoptent des ontologies ou langages différents pour décrire des cas sémantiques. Ces langages ou ontologies varient entre OWL-S, SAWSDL ou un modèle sémantique couvrant un vocabulaire prédéfini. Chacun de ces travaux dispose d'une base de cas qui doit être alimentée au départ par un ensemble de cas sources reflétant des services concrets. Toutefois, les registres de services Web existants utilisent pratiquement tous le standard de description WSDL. Ainsi, la représentation des cas devrait de préférence être compatible avec WSDL. Toutefois, seul le travail de [De Franco Rosa et De Oliviera, 2008] utilise le standard SAWSDL qui permet l'annotation sémantique des services WSDL. Nous constatons aussi que les travaux étudiés ne décrivent pas les cas de façon similaire. Chaque travail procède à définir de sa propre manière de constituer un cas relativement au

langage de description de services Web qu'il adopte, aux besoins de composition ou du domaine d'application qu'il cible.

En plus, tous ces travaux s'intéressent aux propriétés fonctionnelles. Ces propriétés constituent les éléments auxquels est appliqué l'appariement. Notamment, l'information d'ordre fonctionnelle devrait être ciblée en priorité dans la découverte des services Web. Seule la plateforme S-CBR couvre, en plus des propriétés fonctionnelles, les propriétés non-fonctionnelles afin de répondre au mieux aux exigences de ses utilisateurs. Par conséquent, la performance de tout système de découverte à base de RàPC dépend de sa capacité à couvrir le plus de propriétés descriptives des services Web.

Par ailleurs, au niveau des méthodes d'appariement, les travaux étudiés adoptent des techniques d'appariement sémantique fondées sur des mesures de similarité qui intègrent parfois la pondération des attributs appariés. Ceci a pour objectif de donner un poids élevé à un attribut par rapport à un autre lors du calcul de similarité. Par ailleurs, l'appariement mis en place dans le travail de [De Franco Rosa et De Oliviera, 2008] consiste d'abord en une recherche syntaxique dans la base de cas avant d'effectuer une recherche sémantique dans l'UDDI si nécessaire. La recherche syntaxique ne permet pas d'identifier des cas similaires qui ne s'apparient pas syntaxiquement et redirige le système vers une recherche coûteuse dans l'UDDI.

Finalement, l'organisation de la base de cas permet d'effectuer une sélection rapide, et ainsi de pouvoir satisfaire le client en un temps de réponse minimal. Seuls les travaux de [(Osman et al., 2006 a) (Osman et al., 2006 b) (Osman et al., 2009)] et [(Lajmi et al., 2006a) (Lajmi et al., 2006b) (Lajmi et al., 2007) (Lajmi et al., 2011)] prennent cet aspect en considération et proposent respectivement un partitionnement de la base de cas par caractéristiques exprimées selon un vocabulaire prédéfini ou par domaine métier.
En effet, l'organisation par caractéristique permet de chercher dans une partition contenant des cas ayant une même caractéristique. Cependant, la question qui se pose est : ce partitionnement est-il adéquat dans le cas de services Web ? Par exemple, lorsque la requête cherche un service ayant une catégorie métier donnée et offrant une fonctionnalité spécifique, chercher dans une partition de services qui partagent la même catégorie métier n'est pas la manière la plus adéquate. Ceci peut être expliqué par le fait que cette partition peut contenir

une multitude de services qui offrent des fonctionnalités autres que celle demandée même s'ils appartiennent à la même catégorie que celle spécifiée.
D'autre part, l'organisation par domaine métier résout partiellement le problème de rapidité de sélection, du fait que l'application d'une telle organisation ne sera bénéfique que dans une base de cas couvrant plusieurs domaines métiers. Toutefois, une organisation hiérarchisée sera aussi indispensable pour faciliter la recherche du cas dans une base de cas relative à un même domaine métier.

3.4. Conclusion

Le besoin continu d'améliorer la découverte des services Web a encouragé le développement de plusieurs approches. L'approche hybride combine les techniques des approches algébrique et déductive. Cette approche comporte une catégorie de travaux qui adoptent les raisonnements de l'intelligence artificielle pour guider une découverte de services Web dynamique tout en profitant du retour sur l'expérience.

Chacun des travaux de cette catégorie a ses avantages et ses limites. Définitivement, aucun d'eux ne répond efficacement à l'ensemble des critères que nous avons établis dans notre étude. Aussi, toute nouvelle approche de découverte des services Web, quelle que soit la vision sur laquelle elle est fondée, doit-elle, à la fois, profiter des avantages de ses prédécesseurs et combler leurs lacunes en termes de réponse aux critères cités auparavant.

Notre proposition s'intégrera dans le contexte d'une approche fondée sur les principes du RàPC pour la découverte des services Web. Nous souhaitons exploiter ce paradigme en alignement avec les standards W3C des services Web, tout en visant, d'une part, à optimiser le temps de leur découverte, et d'autre part, à améliorer la qualité des résultats, en permettant la découverte de services appropriés répondant aux besoins fonctionnels du client mais aussi à ses préférences et ses exigences particulières. Ainsi, nous envisageons de définir une architecture globale de cette plateforme, d'identifier et de cadrer ses composants. En particulier, l'idée est de définir des mécanismes d'organisation et d'exploitation de la base de cas de cette plateforme, en veillant à optimiser le processus de recherche des cas similaires, et de proposer des techniques de sélection des cas les plus appropriés.

Deuxième partie

Contributions

Chapitre 4

L'approche CBR4WSD

SOMMAIRE

4.1. Critères et mise en perspective

La découverte automatique de services Web a reçu beaucoup d'attention de la part des chercheurs aussi bien que des industriels. En effet, l'approche syntaxique que la norme UDDI (Universal Description Discovery and Integration) fournit pour la découverte de services Web ne répond pas aux besoins d'une découverte et sélection automatique.

C'est dans ce contexte que les approches proposées dans la littérature focalisent sur l'utilisation d'un processus d'appariement (matching) sémantique pour la découverte automatique des services Web. Par conséquent, elles intègrent la sémantique dans la description des services pour en assurer une meilleure interprétation et par la suite assurer une découverte efficace par l'identification et la sélection des services adéquats.

La prospection des travaux dans ce domaine nous a permis de mettre en évidence un certain nombre de critères que nous considérons indispensables pour l'orientation de toute contribution possible dans ce sens. Ces critères peuvent contribuer à l'amélioration de la découverte des services Web et se résument dans les quatre points suivants : « expressivité de description », « alignement avec les standards », « rationalisation du traitement » et « contrôle et maîtrise de la volumétrie ».

Expressivité de description

L'absence de la prise en considération de sémantique des services Web dans le standard WSDL ne favorise pas leur découverte automatique. Aussi, plusieurs modèles sémantiques de description des services Web ont-ils été proposés dans la littérature (cf. chapitre 2).

Toutefois, un modèle de description des services Web expressif, en termes d'intégration de l'aspect sémantique, mais aussi de couverture des propriétés aussi bien fonctionnelles que non fonctionnelles des services Web, demeure fondamental pour le processus de découverte automatique. Ce modèle contribue à l'efficience du processus de découverte puisqu'il permet d'aboutir à des résultats pertinents en termes de réponse aux besoins fonctionnels attendus, mais aussi de réponse aux exigences en préférences non-fonctionnelles.

Alignement avec les standards

Le critère d'alignement avec les standards trouve sa justification dans la tendance qu'a l'industrie informatique et qui consiste à opter pour les normes. Dans le cas de la découverte des services Web, le processus d'appariement explore des éléments utilisés pour la

description. Il convient alors que son approche soit alignée avec les normes recommandées de description des services Web, à savoir les standards W3C.
Toute approche élaborée dans ce sens serait en mesure, d'une part, d'explorer les services Web disponibles sur Internet et décrits selon les standards, et d'autre part, d'utiliser si nécessaire des technologies ou outils existants sur le marché, qui sont pour la plupart d'entre eux, alignés avec les standards W3C.

Rationalisation du traitement

La découverte automatique de services Web est généralement fondée sur leur appariement. Cette opération s'avère énormément coûteuse en termes de traitement vu le nombre élevé de services Web disponibles sur Internet. Aussi, toute approche de découverte de services Web devrait-elle être rationnelle en termes de traitement. Dans ce contexte, nous pensons qu'une approche efficace devrait éviter de dupliquer des traitements onéreux. Il convient donc que cette approche soit fondée sur la mutualisation du traitement de manière à ce que les résultats d'un traitement spécifique puissent être partagés entre des requêtes similaires. Or, la majorité des approches de découverte des services Web procèdent à l'appariement de chaque requête reçue avec l'ensemble des services publiés dans un annuaire UDDI.

Contrôle et maîtrise de la volumétrie

Le processus de découverte de services Web risque d'être un processus coûteux en termes de temps d'exécution vu le nombre élevé de services sur Internet. Aussi, toute approche élaborée dans ce sens doit être capable d'assurer le contrôle et la maîtrise de la volumétrie des services de manière à optimiser le temps de recherche et de traitement. Dans ce contexte, des travaux ont proposé des approches d'indexation syntaxique des registres de services Web. Cependant, ces approches présentent des limites et ne contribuent pas fortement à endiguer le problème de volumétrie. Aussi, pensons-nous qu'une approche efficace de découverte de services Web devrait proposer un mécanisme d'organisation des services Web de manière à optimiser le temps d'accès à ces derniers mais aussi le temps nécessaire pour leur traitement.

Ces quatre critères constituent les principales motivations de cette thèse. Ils évoquent des pistes d'amélioration possibles dans le domaine de la découverte de services Web. Ces pistes sont des axes auxquels notre approche contribue à travers un ensemble de mécanismes fondamentaux.

4.2. Mécanismes fondamentaux

Les orientations de notre approche sont concrétisées par trois mécanismes :
Elle propose un modèle sémantique aligné avec les standards pour assurer une description expressive et conforme aux standards de services Web. Elle opte pour le Raisonnement à Partir de Cas pour la rationalisation du traitement. Enfin, elle propose une organisation à base de communautés de services Web pour assurer le contrôle et la maitrise de leur volumétrie [EL Bitar et al., 2014b].

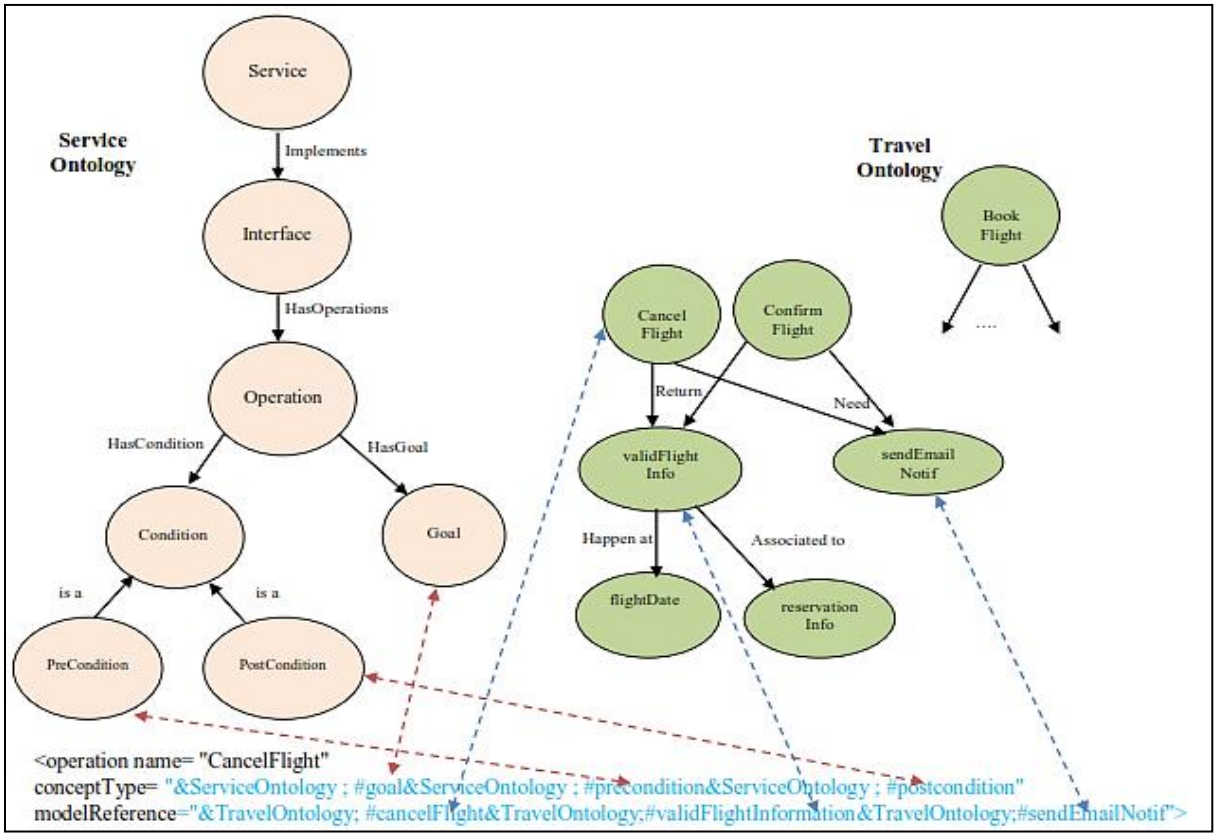
4.2.1. Modèle sémantique aligné avec les standards W3C

Pour répondre au critère d'expressivité, notre approche de découverte de services Web est fondée sur un modèle sémantique aligné avec les standards W3C : WSDL 2.0, SAWSDL et WS-Policy 1.5 [El Bitar et al., 2014a].

L'intégration de la sémantique dans la description des services Web a permis de faire évoluer les mécanismes de découverte au-delà des mécanismes syntaxiques habituels. Plusieurs approches ont proposé des modèles sémantiques pour la description des services Web à savoir OWL-S, SAWSDL et WSMO. Ces modèles décrivent les services Web par des concepts sémantiques issus des ontologies.
Cependant, OWL-S et WSMO sont des produits de recherche qui n'ont pas été standardisés et en plus ils sont dépendants d'un langage spécifique d'ontologie : OWL pour OWL-S et WSML pour WSMO. C'est dans ce contexte que nous avons opté pour le standard SAWSDL qui définit un mécanisme d'annotation des éléments de WSDL à l'aide d'une ou plusieurs ontologies et indépendamment de tout langage de représentation sémantique (WSML, OWL etc.).

Notre modèle sémantique de description de services Web permet, notamment, d'annoter une opération d'un service WSDL par une liste de concepts sémantiques tels que son but (goal), ses post-conditions et ses préconditions. Toutefois, dans le but de faciliter le processus d'appariement qui a besoin d'identifier les éléments de cette liste en tant que tels, nous adoptons le moyen d'annotation de SAWSDL proposé par Chabeb et Aljoumaa [Chabeb and Tata, 2008], [Aljoumaa et al., 2010]. Cette extension demeure alignée avec le standard SAWSDL qui est extensible par définition, et en plus, n'impose aucune contrainte d'adaptation dans les plateformes utilisateurs.

Elle consiste à introduire un nouvel attribut que nous appelons « ConceptType » pour spécifier les types des concepts qui correspondent, un à un dans l'ordre de leur déclaration, à des concepts d'ontologie référencés dans l'attribut "modelReference" de SAWSDL.



Comme illustré dans la figure 4.1, l'attribut "modelReference" annote l'opération "AnnulerRéservation" par trois concepts de l'ontologie du domaine Voyage (Travel Ontology): "cancelFlight", "validFlightInfo" et "sendEmailNotif". L'attribut "conceptType" permet de distinguer la nature de l'information apportée par ces trois concepts en indiquant qu'ils correspondent respectivement aux concepts de l'ontologie de service (Service Ontology): "goal", "precondition" et "postcondition".

Par ailleurs, outre la couverture des propriétés fonctionnelles et de l'aspect sémantique des services Web, notre modèle couvre également les propriétés non fonctionnelles conformément au standard WS-Policy. Ce dernier permet de définir des politiques de services Web en termes d'exigences, préférences et caractéristiques générales. Il permet, notamment, aux fournisseurs aussi bien qu'aux consommateurs de services Web d'exprimer des exigences ou préférences en termes de sécurité ou de qualité de service (QoS). Par exemple, la figure 4.2 illustre un

extrait de la description WS-Policy correspondante à l'opération "bookFlight". Dans cette description nous trouvons des contraintes classiques qui ne sont pas en rapport avec les QoS tels que le type d'encodage (ligne 10) et la langue utilisée (ligne11). Ces contraintes seront exprimées conformément aux normes de WS-Policy.

```
1  <wsp:Policy
2      xmlns:wsp="..."
3      xmlns:wsse="..."
4      xmlns:qoso=".../ontology/operator"
5      xmlns:qosu=".../ontology/unit"
6      ...>
7
8      <wsp:ExactlyOne>
9          <wsp:All>
10         <wsp:TextEncoding wsp:Usage="wsp:Required" Encoding="Utf-8"/>
11         <wsp:Language wsp:Usage="wsp:Optional" Language="English" wsp:Preference="1"/>
12         <wsp:Assertion name="RTassertion" ...>
26     </wsp:ExactlyOne>
27 </wsp:Policy>
```

Figure 4.2 : Extrait de la description non fonctionnelle associée à l'opération "bookFlight" du service de gestion de vols.

4.2.2. Raisonnement à Partir de Cas

Dans l'optique de rationalisation du traitement de découverte de services Web, notre approche est fondée sur le retour d'expérience, notamment sur les mécanismes du RàPC (Raisonnement à Partir de Cas). Cela devrait lui permettre justement de capitaliser sur l'expérience pour livrer des solutions inspirées des cas similaires déjà traités, et aussi d'en sélectionner les meilleurs sur la base de l'expérience de leur exécution.

Le choix du RàPC est justifié par le fait que les systèmes de réutilisation antérieurs et traditionnels qui sont des systèmes à base des règles présentent beaucoup de limites, tels que les difficultés dans l'acquisition des connaissances, l'incapacité de capitalisation sur l'expérience à travers une mémoire de résolution de problèmes, l'inefficacité d'inférence et de traitement des exceptions et la performance de l'ensemble du système qui laisse à désirer. [El Bitar, 2010].

Or, un système RàPC est un système à base de connaissance qui consiste à collecter, représenter et mémoriser des expériences (cas), au lieu de recenser un ensemble de règles qui risquent d'être non-exhaustif et donc peu fiable. En outre, il est fondé sur le calcul de similarité entre les cas et leur adaptation qui constituent, en comparaison avec les systèmes à base de règles, une solution au traitement des cas où les connaissances s'avèrent insuffisantes

ou incomplètes. Un système RàPC est également capable de s'autoalimenter par de nouveaux cas pour produire des solutions raffinées. Dans le cadre de la découverte de services Web, le système RàPC peut notamment démarrer avec un nombre limité de cas disponibles et enrichir progressivement sa base de cas au fur et à mesure du traitement des requêtes reçues.

4.2.3. Communautés de services

Dans l'optique de contribuer au contrôle et à la maitrise de la volumétrie des services Web, en particulier des cas disponibles qui risquent de se multiplier au fur et à mesure de la réutilisation des problèmes, notre approche est fondée sur l'organisation de la base de cas de manière à optimiser le processus de découverte. Alors que cet aspect demeure important, nous avons constaté que les travaux de découverte de services Web à base de RàPC ne s'intéressent pas à l'organisation de la base de cas à l'exception des travaux de [(Osman et al., 2006 a) (Osman et al., 2006 b)] et [(Lajmi et al., 2006a) (Lajmi et al., 2006b)] qui proposent respectivement un partitionnement de la base de cas par caractéristiques (exprimées selon un vocabulaire prédéfini) ou par domaine métier. Toutefois, ces méthodes d'organisation permettent de résoudre partiellement le problème d'optimisation du processus de recherche des cas similaires (cf. chapitre 3, section 3.3.5).

C'est dans ce contexte que nous proposons d'organiser la base de cas par communautés de services dans le but de garantir une remémoration efficace et plus rapide, en réduisant le champ de recherche aux cas sources adéquats, et par la suite, réduire la tâche du processus d'appariement. Nous désignons par communauté de services, l'ensemble des services Web qui sont similaires de point de vue fonctionnel [Belouadha, 2007]. Aussi, proposons-nous d'organiser la base de cas par communautés de services qui regroupent chacune des services (solutions) répondant à des problèmes qui expriment des besoins fonctionnels similaires, indépendamment des exigences ou préférences non-fonctionnelles qu'ils traduisent.
L'objectif est de pouvoir localiser la communauté de services, par conséquent l'ensemble de cas qui doivent être explorés à la recherche d'une solution à un problème cible donné. Le fait de focaliser sur l'aspect fonctionnel va de soi puisque l'objectif de tout client serait naturellement de trouver une solution qui doit obligatoirement satisfaire son besoin fonctionnel, et qui répond dans la mesure du possible à ses exigences ou préférences non-fonctionnelles. Par ailleurs, au fur et à mesure de la résolution de problèmes, les communautés de services peuvent s'enrichir par de nouveaux cas.

4.3. Processus de traitement

4.3.1. Architecture fonctionnelle

L'approche CBR4WSD que nous proposons est fondée sur cinq couches (cf. figure 4.3) de traitement qui, à l'aide d'ontologies, prennent en compte l'aspect sémantique des services Web, pour pouvoir repérer parmi les cas disponibles, des cas similaires à un cas cible, en proposer une meilleure solution, et enrichir la base de cas par le ou les nouveaux cas identifiés et dont le test a été validé.

Une couche transversale est également considérée pour assurer l'administration du système i.e. l'acquisition et la mise à jour des connaissances de base, telles que la base de cas et les ontologies utilisées pour le traitement sémantique.

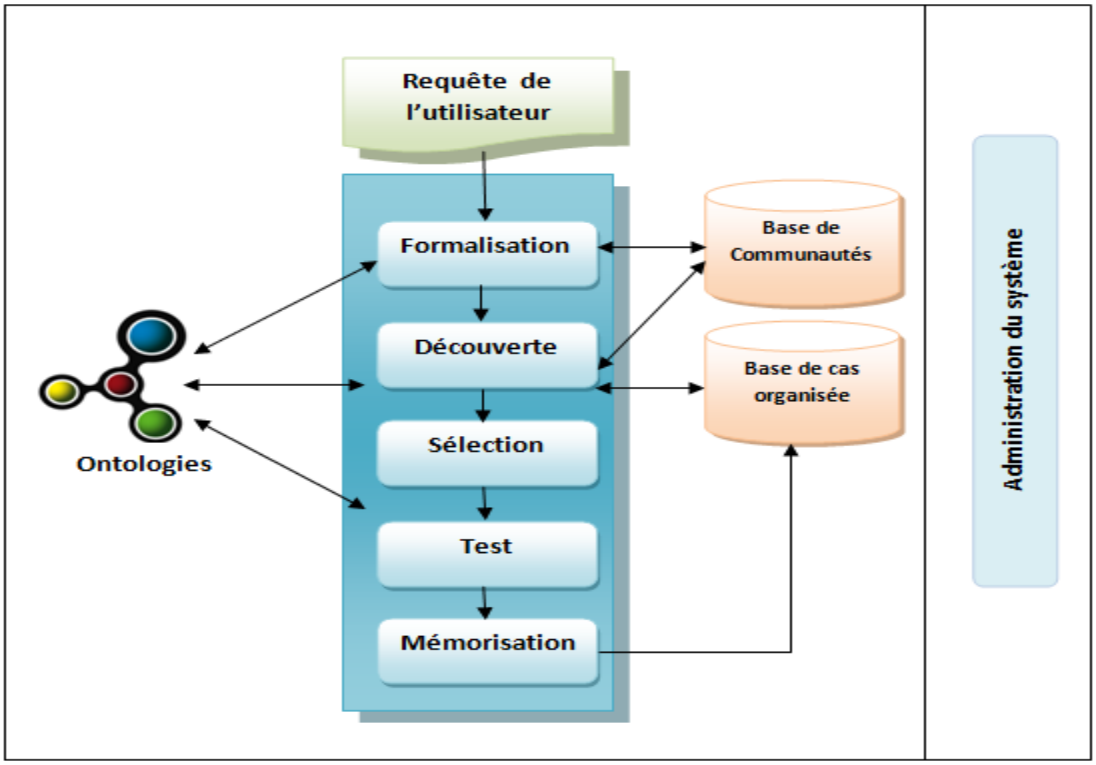


Figure 4.3: Vue globale de CBR4WSD.

Les cinq couches principales de traitement consistent en la formalisation de la requête du client, la découverte des services pouvant y répondre, la sélection du ou des services les plus adéquats, le test de ces services et la mémorisation des nouveaux cas identifiés et validés pour l'enrichissement de la base de cas.

La figure 4.4 établit le rapprochement entre le processus CBR4WSD et le cycle de référence RàPC.

Couche « Formalisation »

La couche « Formalisation » est la première couche du traitement dont le but est de représenter la requête sous un format devant permettre son traitement automatique. Elle consiste à exprimer la requête qui constitue la partie problème du cas cible, sous forme d'un ensemble de descripteurs conformément aux principes du RàPC. Dans le cas de notre approche, ces descripteurs sont répartis en descripteurs fonctionnels et non-fonctionnels qui traduisent respectivement les exigences fonctionnelles du client et ses préférences non-fonctionnelles. La couche « Formalisation » correspond à l'étape « Elaboration » qui consiste à formaliser la description du problème à résoudre en attribuant des valeurs aux descripteurs correspondants du cas cible (cf. figure 4.4).

Notre approche propose à l'utilisateur une interface sous forme de « Template » constituée de champs à remplir pour exprimer ses besoins. Les informations saisies sont ensuite récupérées pour construire un cas cible tout en intégrant leur sémantique. Il est à noter que les requêtes des clients peuvent être décrites par différents langages [El Bitar et al., 2011a] . Elles peuvent être exprimées de manière formelle au moyen de graphes, de la logique temporelle ou la logique de premier ordre à titre d'exemples, comme elles peuvent aussi être exprimées en langage naturel puis transformées pour être représentées dans un langage formel. De nombreux travaux qui s'inscrivent dans ce cadre ont été proposés dans la littérature. Aussi, cet aspect de traitement des requêtes exprimées en langage naturel ne fait-il pas l'objet de ce travail.

Couche « Découverte »

La deuxième couche dite « Découverte » reprend le cas cible formalisé par la couche « Formalisation » pour le comparer avec les cas disponibles. Elle cherche les services pouvant y répondre parmi ceux figurant dans ces cas sources. Comme illustré dans la figure 4.4, cette couche correspond à une sous-étape de la deuxième phase du cycle RàPC dite « Remémoration » dont l'objectif est de trouver des cas sources similaires à un problème cible pour sa résolution.

Elle consiste dans le cas de notre approche à explorer une base de cas organisée par communautés de services pour découvrir ceux qui sont similaires au cas cible du point de vue fonctionnel. Cela permettra comme nous l'avons mentionné auparavant, d'optimiser le temps de découverte en ne cherchant que dans les cas associés à la communauté de service adéquate.

Un algorithme d'appariement sémantique est utilisé pour calculer la mesure de similarité fonctionnelle (MSF) consistant en l'agrégation des similarités locales effectuées sur les descripteurs fonctionnels des cas.

Couche « Sélection »

La couche « Sélection » correspond comme illustré dans la figure 4.4, à la sous étape de sélection de la phase « Remémoration » du cycle RàPC. Elle considère les résultats de la couche « Découverte » pour en sélectionner le meilleur cas source fonctionnellement similaire au cas cible. A ce stade, une mesure de similarité non-fonctionnelle (MSNF) est calculée. Cette mesure est l'agrégation des similarités locales effectuées sur les descripteurs non-fonctionnels des cas. Elle est elle-même agrégée à la mesure de similarité fonctionnelle (MSF) pour générer une mesure de similarité globale (MSG) permettant d'identifier le ou les services les plus appropriés, capables de mieux répondre aux besoins fonctionnels aussi bien que non-fonctionnels exprimés au niveau du cas cible.

Couche « Test »

Concernant la phase de révision, elle consiste à tester la solution adaptée afin de vérifier si elle convient pour résoudre le problème, avant de décider de la mémoriser. La couche qui correspond à cette phase est prise en considération dans notre approche sauf qu'elle est confiée au client qui peut exprimer sa satisfaction ou non vis-à-vis le service. Selon que le degré de satisfaction du client est supérieur ou inférieur à un seuil défini par l'administrateur du système de découverte, le nouveau cas identifié sera mémorisé ou non dans la base de cas. Cette phase contribue également au processus de mise à jour et de nettoyage de la base de cas. Le client ayant effectué le test, peut notamment détecter que le service proposé a été désactivé ou a subi des changements par son fournisseur, et en informer l'administrateur du système. Ce dernier devra par la suite procéder à la mise à jour ou carrément à la suppression des cas concernés.

Couche « Mémorisation »

La couche « Mémorisation », comme son nom l'indique, correspond à l'étape de « Mémorisation » du cycle RàPC. Elle consiste en la mise à jour de la base de cas par l'introduction d'un cas cible résolu si ce dernier n'est pas équivalent, mais plutôt similaire à

un cas source existant. L'insertion du nouveau cas concerné doit être opérée au niveau de sa communauté de services.

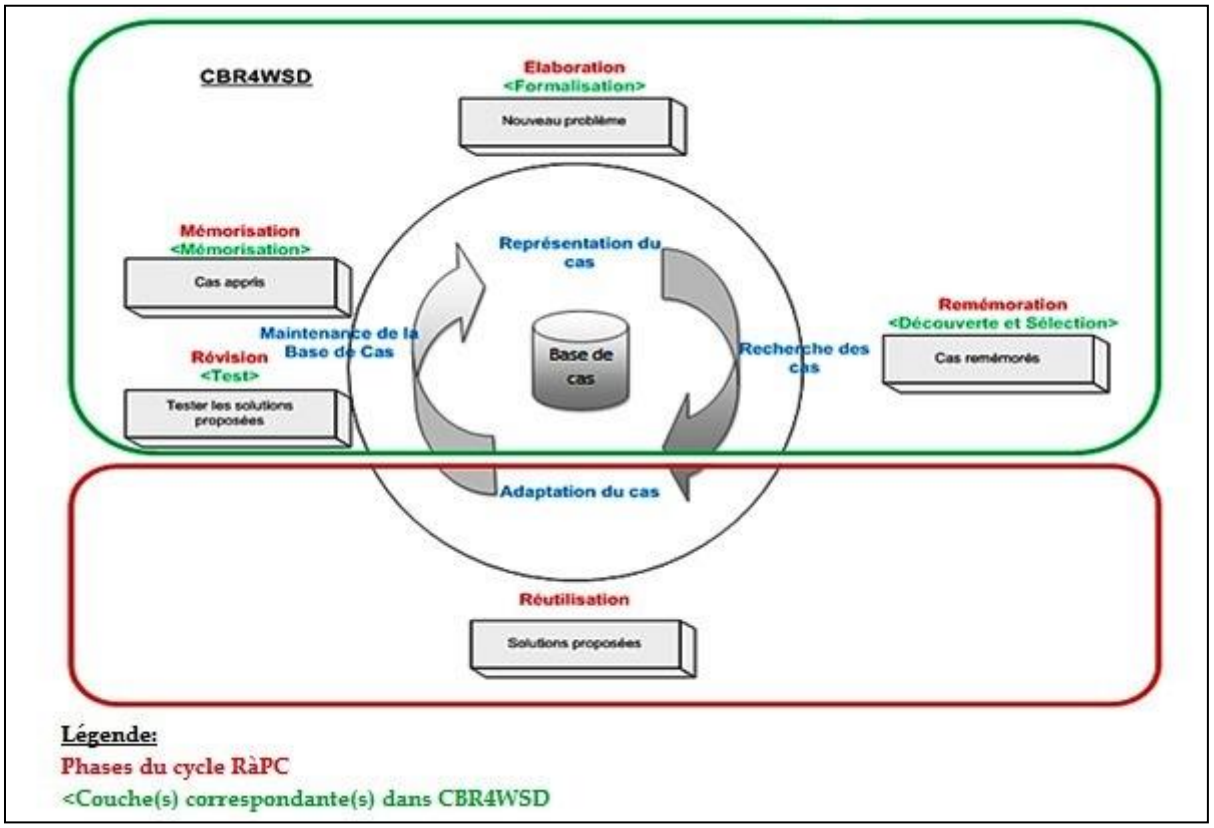


Figure 4.4 : Positionnement de notre approche dans le cycle RàPC.

La phase de réutilisation du cycle RàPC consiste à adapter une solution d'un cas source remémoré pour résoudre le problème cible si, parmi les descripteurs de cette solution, il existe quelques-uns qui ne sont pas tout à fait conformes aux besoins du client. Or, dans le cas spécifique de découverte de services Web, l'adaptation d'un service disponible n'est pas du ressort d'un système dédié à la découverte. Elle ne doit être envisagée que dans le cadre d'un processus de maintenance et d'adaptation de services ; celui-ci est déclenché par un fournisseur de services souhaitant les améliorer pour mieux répondre aux besoins des clients.

Finalement, en se référant au cycle RàPC, nous constatons que notre approche CBR4WSD couvre les phases « Elaboration », « Remémoration », « Révision » et « Mémorisation » du cycle RàPC. Le fait de ne pas prendre en considération la phase de « Réutilisation » se justifie par la spécificité du domaine d'application qui est la découverte de services Web. Après avoir décrit les grandes lignes de notre approche, nous présentons par la suite le processus global de découverte qui consiste en deux processus offline et online.

4.3.2. Processus offline

A l'instar du modèle proposé par Lamontagne et Lapalme [Lamontagne et Lapalme, 2002], notre modèle de découverte de services Web est fondé sur deux processus, un processus online et un processus offline, adaptés aux spécificités de l'approche CBR4WSD (cf. figure 4.5).

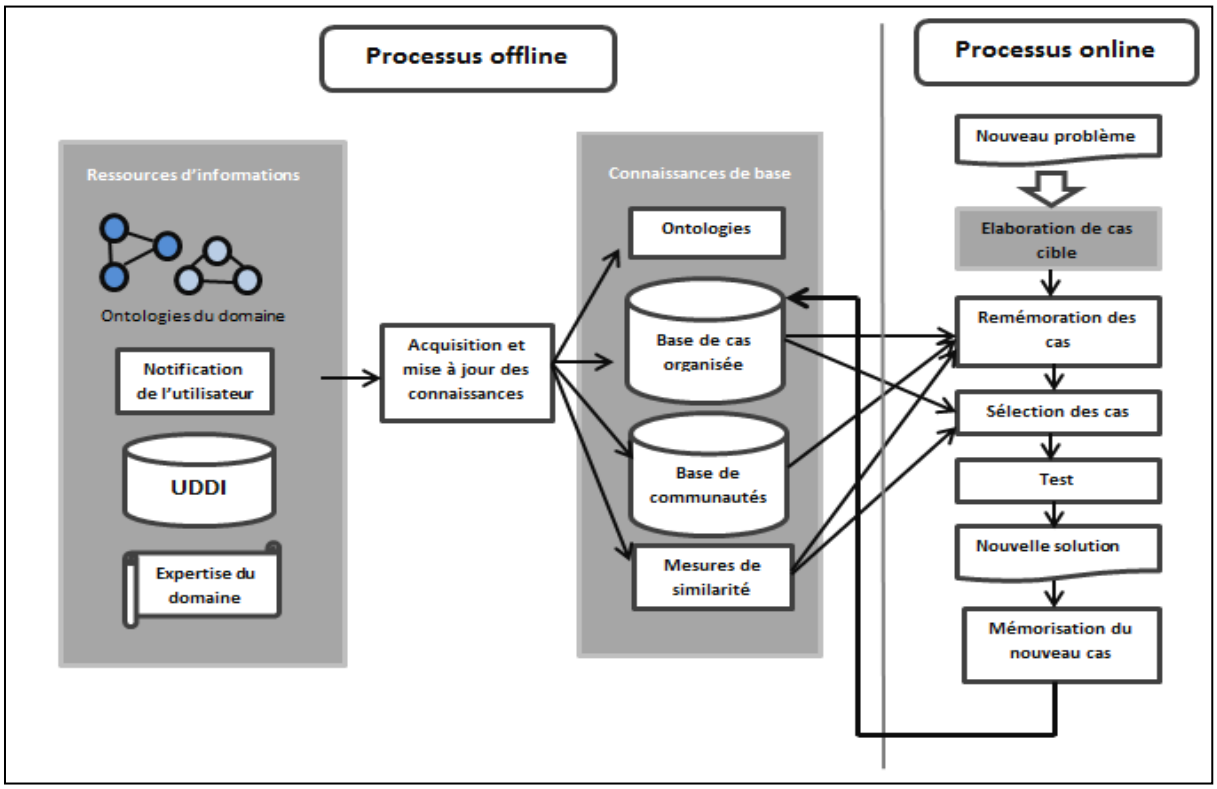


Figure 4.5: Modèle du processus global de l'approche CBR4WSD.

Le processus offline est un processus de configuration et d'administration du système. Il constitue un processus d'acquisition et de mise à jour de connaissances indispensables pour l'utilisation du système. Déclenché par l'administrateur du système, ce processus permet de construire ou spécifier les liens d'ontologies à utiliser, construire la base de cas, construire la base de communautés et lancer le calcul de similarités entre les différents concepts sémantiques mis en jeu dans le système.

L'administration peut puiser ces connaissances à partir des ontologies de domaine existantes et des registres UDDI, tout en s'appuyant aussi sur son expertise dans le domaine. Le processus d'acquisition des connaissances (processus offline) se poursuit également, durant toute la durée de vie du système, par la mise à jour de ces connaissances chaque fois que cela s'avère nécessaire (suite aux nouveautés ou changements dans le domaine détectés par l'administrateur ou qui lui sont notifiés par les utilisateurs du système).

Par ailleurs, le choix de calcul de similarités est un processus coûteux en termes de temps. Le choix de l'effectuer en mode offline se justifie par l'objectif d'optimisation du processus de découverte online. Cela devrait permettre notamment de minimiser considérablement le temps réservé à l'appariement des cas sources au cas cible. Cette manière de faire permet donc d'éviter de dupliquer le calcul de similarité entre un concept sémantique du cas cible et un autre qui figure dans plusieurs cas sources. L'idée est que le calcul de similarité entre deux concepts sémantiques ne soit effectuée qu'une seule fois indépendamment des requêtes et aussi des cas sources traités pour chaque requête.

Finalement, les connaissances constituées par le processus offline forment une ressource d'information pour le processus online qui contribue également à l'enrichissement, notamment, à travers l'alimentation de la base de cas découverts. Les détails sur ce processus sont présentés dans la section suivante.

4.3.3. Processus de découverte online

Le processus online de notre approche CBR4WSD est déclenché à la réception d'une requête de recherche de services Web (cf. figure 4.6 étape 1). Ce processus démarre par la transformation des données de la requête récupérées à travers de la « Template » remplie par le client, en un cas cible aligné avec les standards W3C : SAWSDL et WS-Policy 1.5 (cf. figure 4.6 étape 2). Cette opération, assurée par un générateur de cas cible sémantique, consiste à représenter la requête sous forme d'un cas décrit par un ensemble de descripteurs sémantiques du problème, répartis en descripteurs fonctionnels et non-fonctionnels, en plus des descripteurs de la solution que le processus Online tentera d'instancier par un service Web découvert. Autrement dit, les descripteurs du problème représentent un service Web abstrait, alors que les descripteurs de solution représentent le service Web concret correspondant.

L'opération de génération du cas cible sera suivie de son élaboration dans le but de terminer l'étape de formalisation de ce cas (cf. figure 4.6 étape 3). Notamment, le composant élaborateur de cas cible d'un moteur RàPC du système, est chargé de compléter la description du cas cible en l'annotant par la communauté de services à laquelle il correspond. Il doit, à l'aide des descripteurs fonctionnels du problème cible, identifier à partir de la base de communautés, celle à laquelle est associé le cas cible. Les détails sur la formalisation des cas seront présentés dans le chapitre suivant.

Par ailleurs, le processus global de découverte se poursuit par la remémoration des cas sources similaires au cas cible (cf. figure 4.6 étape 4). A ce stade, le composant dit « Matchmaker sémantique Online » procède, après avoir identifié les cas sources dont la communauté de services associée est équivalente à celle associée au cas cible, au calcul de leur similarité fonctionnelle (MSF) avec le cas cible. A l'achèvement de cette opération, un ensemble de cas sources dont la mesure de similarité calculée satisfait un seuil défini sera remémoré.

L'ensemble des cas sources remémorés est ensuite mis à la disposition d'un sélectionneur chargé d'en identifier le meilleur cas relativement aux propriétés non-fonctionnelles demandées (cf. figure 4.6 étape 5). Ce composant est en mesure d'effectuer un « matching » appliqué aux politiques sources et cible, et de calculer par la suite, les mesures de similarité globale (MSG) entre les cas sources et le cas cible. Le but étant d'identifier les services Web à recommander au client, ces services sont les solutions associées aux cas sources dont la mesure de similarité globale (MSG) est la plus élevée.

Il est à noter que lors du calcul de similarités, aussi bien fonctionnelles que non-fonctionnelles, entre les cas sources et le cas cible, le traitement de l'incomplétude de connaissances, c'est à dire à la non instanciation de certains descripteurs du problème est prévu dans notre approche. Ce traitement est discuté au niveau du chapitre 5.

Finalement, après avoir reçu la notification sur le choix du client, seul le service Web ayant donné satisfaction servira à instancier les descripteurs solution du cas cible. Le cas qui en résulte sera introduit dans la base de cas à l'aide du composant dit « Mémorisation de cas » (cf. figure 4.6 étape 6). Selon le résultat du test de fonctionnement des services de la base de cas, des notifications seront également adressées à l'administrateur du système pour qu'il procède à la mise à jour de la base de cas si une désactivation ou un changement du service ont été détectés.

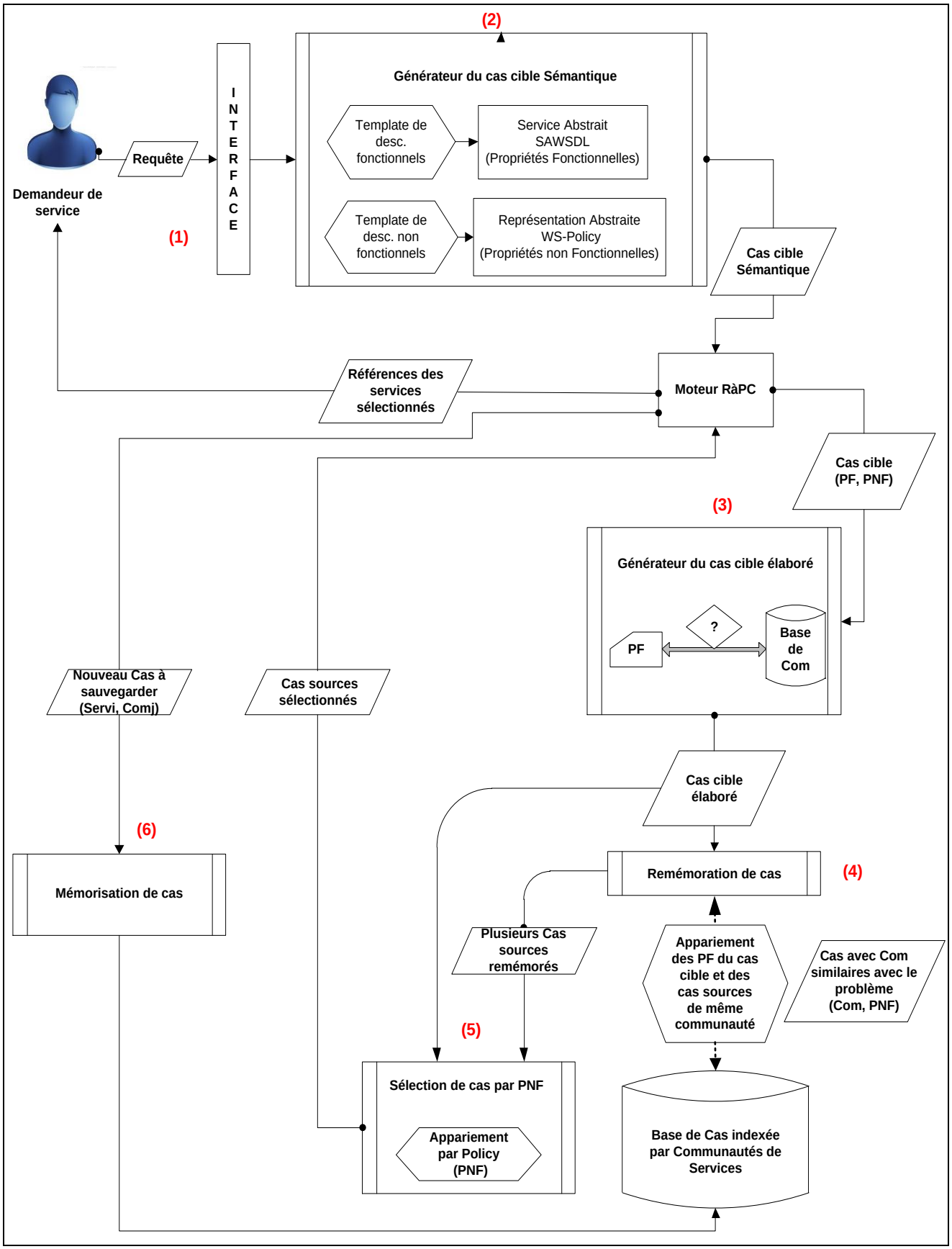


Figure 4.6 : L'approche CBR4WSD.

4.4. Synthèse

Beaucoup de travaux dans la littérature ont proposé des approches pour la découverte de services Web, allant des approches de découverte à partir d'UDDI aux approches de découverte à partir de la base de cas. L'approche CBR4WSD s'inscrit dans cette deuxième catégorie. Elle constitue une approche à base de RàPC pour la découverte de services Web sémantiques. Son contour de contribution inclut un ensemble d'aspects qui visent à remédier aux limites des approches existantes et qui font l'originalité de l'approche CBR4WSD. Ces aspects sont essentiellement relatifs à la rationalisation du traitement, le contrôle et la maîtrise de la volumétrie des services Web traités et le respect des normes, sans oublier l'amélioration de la qualité des résultats obtenus en termes de leur capacité à répondre aux besoins, aussi bien fonctionnels que non-fonctionnels, du client.

Dans ce contexte, l'approche CBR4WSD est fondée sur un modèle sémantique aligné avec les standards W3C pour la description des services Web objets de la découverte. Ce choix devrait élargir le périmètre d'application de notre approche de façon à ce qu'elle ne soit pas uniquement confinée aux laboratoires. Il permet également, à travers l'intégration de la sémantique et la couverture des aspects fonctionnels et non-fonctionnels, de découvrir non seulement les services Web pouvant répondre à la requête client mais d'en sélectionner les meilleurs.

L'approche CBR4WSD adopte également les mécanismes du RàPC et propose une organisation de la base de cas par communautés de services dans le but de rationaliser le traitement et d'optimiser le temps de découverte.

Deux processus sont prévus dans le cadre de notre approche. Un processus online couvre cinq couches de traitement : la formalisation, la découverte, la sélection, le test et la mémorisation. Un autre processus offline correspond à une couche transversale assurant l'administration et la configuration du système de découverte.

Le tableau 4.1 synthétise le cadrage de l'approche CBR4WSD.

	Processus	**Couches**	**Mécanismes et fondamentaux**	**Contribution**	**Originalité**
CBR4WSD	Processus Online	• Formalisation • Découverte • Sélection • Test • Mémorisation	• Modèle sémantique aligné avec W3C • RàPC. • Organisation par communautés de services. • Matching sémantique.	• Respect des normes et qualité des résultats • Rationalisation du traitement. • Optimisation du temps de remémoration. • Qualité des résultats.	• Alignement avec les standards W3C : SAWSDL + WS-Policy. • Cycle adapté à la découverte de services Web. • Introduction de la notion de communautés de services centrées Goal pour la découverte de services Web. • Formule de calcul adapté pour un matching centré sur l'aspect fonctionnel et couvrant l'aspect non fonctionnel.
	Processus Offline	Administrateur	•Expertise •UDDI •Ontologies	• Optimisation du temps de matching	• Calcul préalable de similarité.

Tableau 4.1. Propriétés de l'approche CBR4WSD.

Chapitre 5

Formalisation et élaboration des cas

SOMMAIRE

5.1. Introduction

Ce chapitre se focalise sur la phase d'élaboration, première phase du cycle du RàPC, ainsi que l'ensemble des modélisations nécessaires pour formaliser les cas de notre système CBR4WSD. Cette phase consiste en la formalisation de la description du problème à résoudre et sa mise en forme en vue de la remémoration. Nous identifions alors l'ensemble des descripteurs nécessaires pour formaliser les cas manipulés dans notre système CBR4WSD. C'est dans cet esprit et conformément aux règles du RàPC que nous proposons notre modèle de représentation du cas.

5.2. Définition et notation d'un cas

Rappelons que dans la terminologie du RàPC, un cas représente différents types de connaissances qui peuvent être stockées selon différents formats de représentation. Un cas se compose d'un problème noté « pb » et de la solution à ce problème notée « sol (pb) ». Le cas est alors noté: cas= *(pb, sol (pb)).*

Un « cas source » est un cas qui sert d'inspiration pour résoudre un nouveau problème nommé le « cas cible ». La codification est alors la suivante :

Cas-source = (source, sol (source))

Cas-cible = (cible, sol (cible))

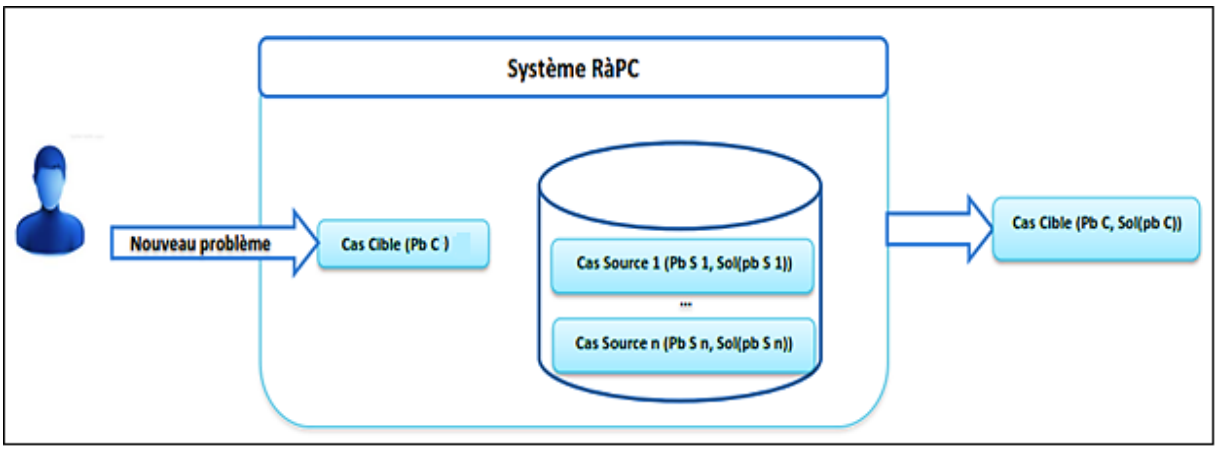


Figure 5.1 : Schéma représentant l'utilisation de Cas Cible et de Cas Sources dans un système RàPC.

Les cas sont décrits par un ensemble de descripteurs qui dépendent du domaine d'application. Un descripteur est une connaissance qui nous permet de décrire le problème. Le descripteur « d » est caractérisé par une paire (a,v) [Gebhardt et al., 1997], où :

- « a » est un attribut défini par un nom,
- « v » est la valeur qui lui est associée.

Un attribut représente une caractéristique du domaine applicatif qui peut être numérique ou symbolique.

Nous pouvons décrire les cas comme suit :

Cas source :

- Source= $\{d_1^S$ ....$d_n^S\}$ où d_i^S est un descripteur du problème source.
- Sol (source)= $\{D_1^S$... $D_n^S\}$ où D_i^S est un descripteur de la solution source.

Cas cible :

- Cible= $\{d_1^C$... $d_n^C\}$ où d_i^C est un descripteur du problème cible.
- Sol (cible)= $\{D_1^C$... $D_n^C\}$ où D_i^C est un descripteur de la solution cible.

Par la suite, nous utilisons ces définitions pour représenter les cas dans notre système CBR4WSD.

CBR4WSD est dédié à la découverte automatique des services Web en réponse à la requête d'un client demandant un service avec des besoins spécifiques.

Ainsi, la partie « *pb* » du cas reflète la requête du client et doit pouvoir décrire ses particularités en termes de besoins fonctionnels et non-fonctionnels. De même, la partie « *sol(pb)* » doit permettre d'identifier les informations pertinentes et suffisantes pour l'invocation du service requis (cf. figure 5.2).

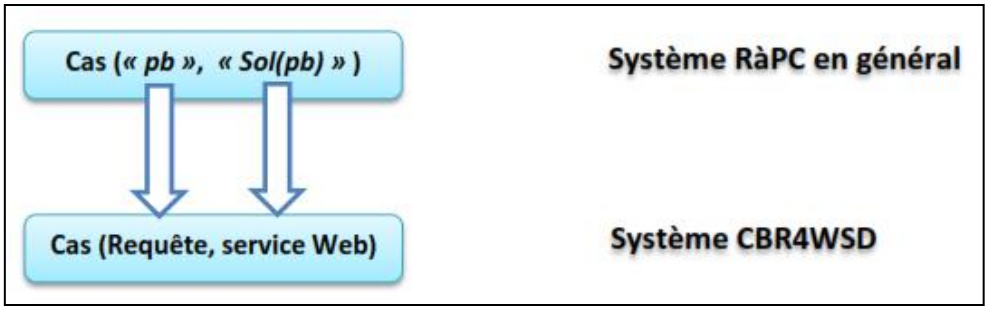


Figure 5.2: Composants d'un cas dans le système CBR4WSD.

5.3. Eléments de description d'un service Web dans CBR4WSD

Les services Web sont décrits généralement en utilisant WSDL, le standard de W3C. Toutefois, le manque de sémantique dans WSDL empêche la découverte automatique de services Web (cf. chapitre 2, paragraphe 1). Pour cette raison, un ensemble de travaux intégrant la sémantique dans la description de leurs services Web a été proposé. Parmi ces travaux, certains considèrent la description des besoins fonctionnels et d'autres les besoins non-fonctionnels aussi, et ce selon différentes approches.

Nous avons choisi dans notre approche, l'alignement avec les standards W3C de description de services Web (cf. chapitre 4). Aussi, étions-nous amenés à formaliser notre cas conformément aux trois standards WSDL 2.0, SAWSDL et WS-Policy. Toutefois, nous avons jugé nécessaire d'intégrer de nouvelles informations fonctionnelles importantes dans la description sémantique de nos services Web. Ainsi, compte tenu de l'extensibilité offerte par SAWSDL, nous l'avons enrichi en incorporant de nouvelles notions telles que le but (Goal), les préconditions et les postconditions, formant ainsi le « SAWSDL étendu » [El Bitar et al., 2014a].

Pour construire notre modèle de description de service Web, nous commençons par identifier les caractéristiques de base prévues par les standards WSDL/SAWSDL.

5.3.1. Eléments de description WSDL/SAWSDL

Dans le standard WSDL, un service Web est constitué d'un ensemble d'opérations assurant chacune une fonction bien déterminée (contrairement à OWL-S qui réduit le service à une seule opération pour faciliter la manipulation). De ce fait, nous considérons que nos services décrits fonctionnellement par le WSDL/SAWSDL sont des ensembles d'opérations. Par ailleurs, pour satisfaire une requête, notre système doit chercher une opération bien spécifique d'un service Web déterminé qui répond au besoin attendu.

Nous présentons dans la figure 5.3 les éléments de base considérés par la description WSDL/SAWSDL.

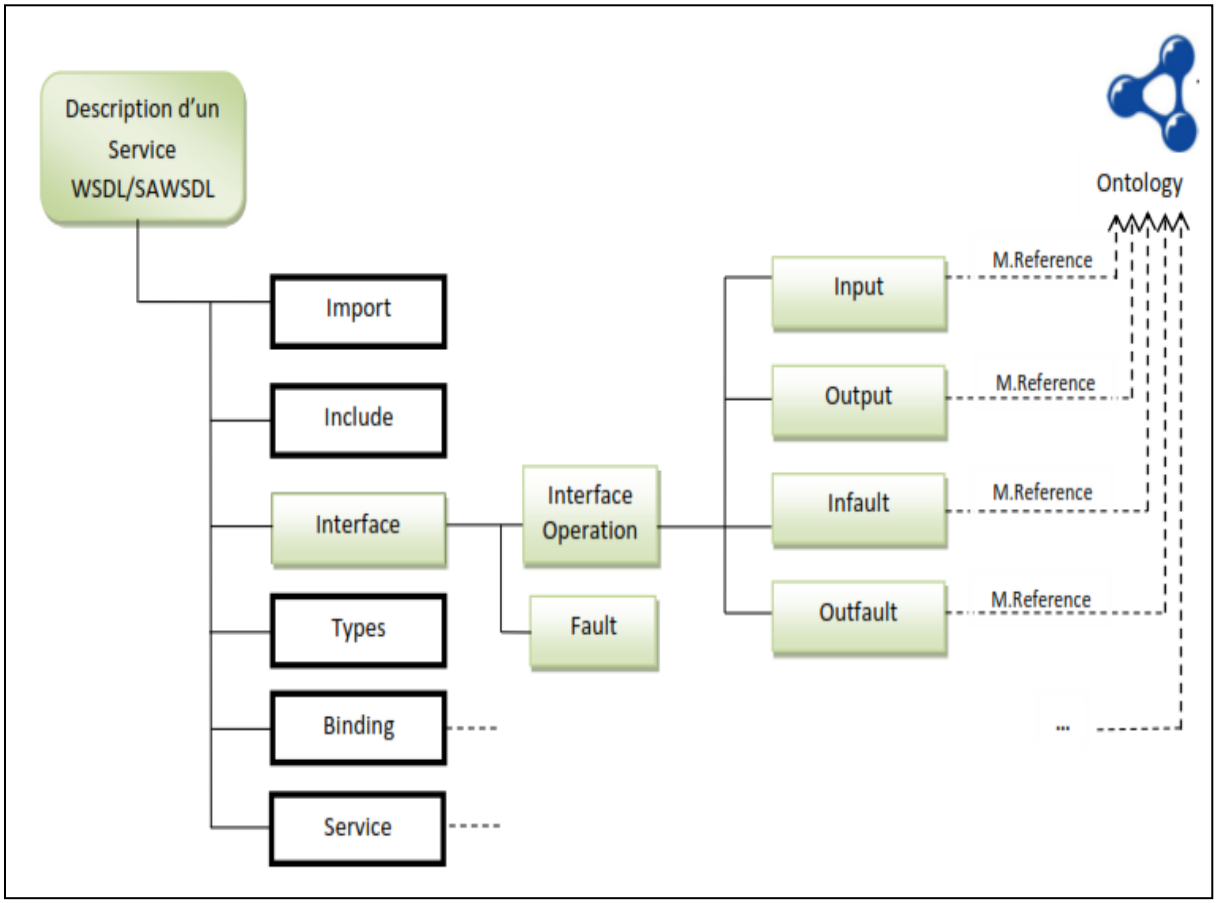


Figure 5.3: Extrait du modèle de la description de base d'un service Web WSDL/SAWSDL

Comme le montre la figure 5.3, une opération dans un service WSDL/SAWSDL sera décrite par des éléments de base tels qu'`Input`, `Output`, `Infault` et `Outfault` pouvant être annotés sémantiquement par le biais de références ontologiques.

5.3.2. Eléments d'extension fonctionnelle

Compte tenu de l'extensibilité de WSDL/SAWSDL, nous l'avons étendu en y ajoutant les éléments `Goal` (but), `precondition` (précondition) et `postcondition` (postcondition) jugées indispensables à la découverte de service Web. Ces trois éléments complètent la description de chaque opération, tout en restant conformes au standard WSDL.

Ces éléments ne sont pas pris en considération dans le langage de description de services Web WSDL, mais ceci n'interdit point cet enrichissement tout en annotant ces éléments par le moyen de référence à des concepts sémantiques d'une ou plusieurs ontologies. La figure 5.4 illustre l'extension que nous proposons.

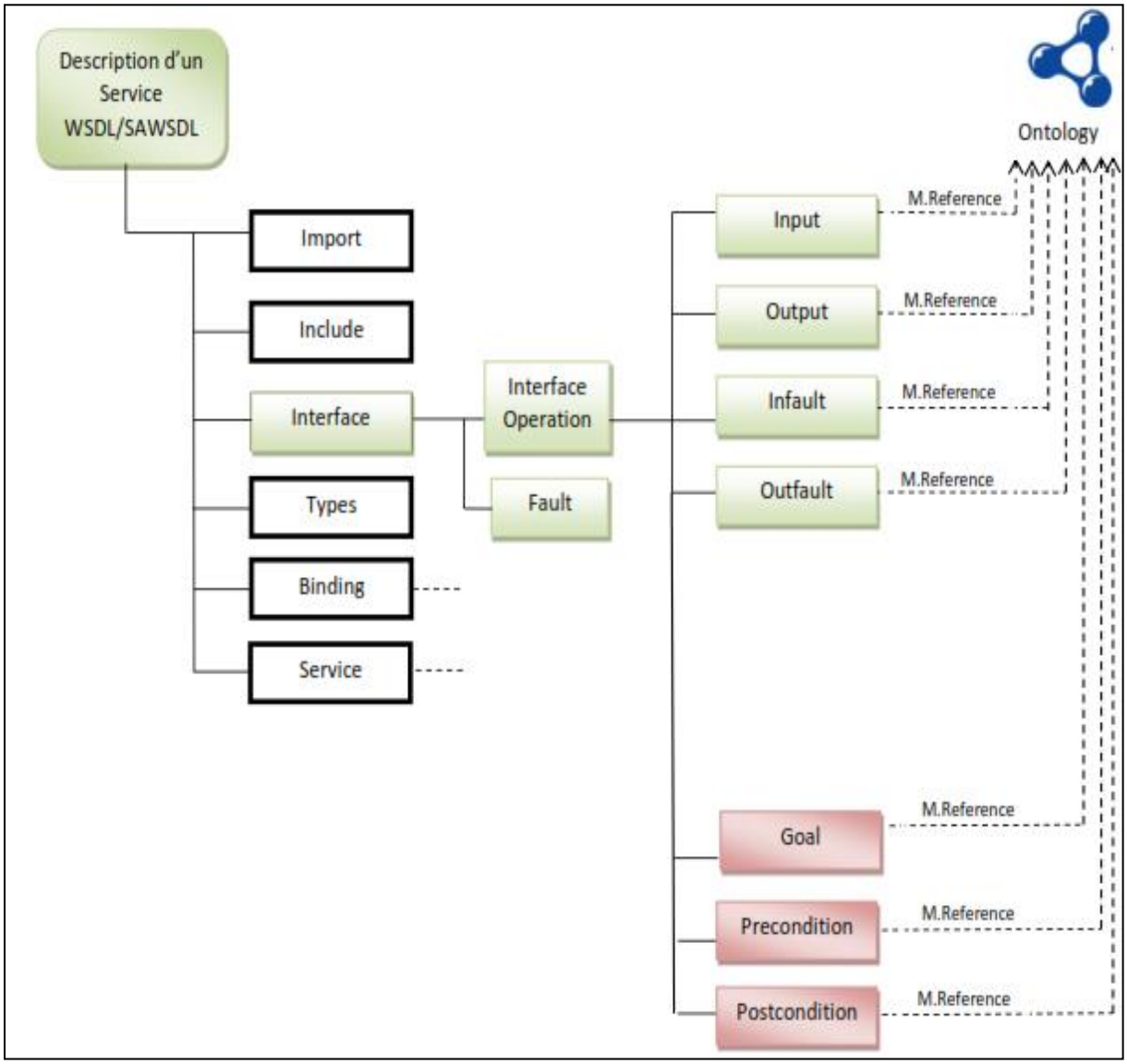


Figure 5.4: Notre modèle de description fonctionnelle enrichie de service Web.

Dans ce qui suit, nous justifions le choix d'intégration de chacun des trois éléments dans la description fonctionnelle du service WSDL/SAWSDL.

L'attribut Goal

Nous estimons que l'attribut `Goal` utilisé dans des approches telles que l'ontologie WSMO est fondamental pour la découverte de services Web, vu qu'il définit l'objectif du client qui cherche ou consulte un service ou une opération d'un service donné. A titre d'exemple, c'est le `Goal` qui distingue les deux services S1 et S2 qui ont les mêmes Inputs et le même Output (cf. tableau 5.1.).

	Inputs	Output	Goal
Service S1	Date, Airline	Liste de Vols	Vols annulés
Service S2	Date, Airline	Liste de Vols	Vols soutenus

Tableau 5.1: Exemple illustrant l'importance de l'attribut "Goal".

Selon le tableau 5.1., le service S1 renseigne sur les vols annulés et le service S2 donne la liste des vols soutenus. Ces deux services auraient été considérés comme services similaires sans l'attribut `Goal` qui indique qu'ils ont deux objectifs différents. Ceci illustre l'importance de l'intégration de la notion du `Goal` dans notre description des services, pour mener à bien leur découverte.

Les préconditions

La prise en considération des `preconditions` permet d'améliorer la pertinence de la découverte en prenant en compte les exigences spécifiques des clients. A titre d'exemple, un service de réservation de vol en ligne permettant un paiement électronique uniquement via une carte bancaire « Express », ne doit pas être sélectionné si le client cherche un service acceptant une carte « VISA ». Il serait incapable de l'utiliser même si ce service répond aux besoins de réservation de vol.

L'intégration de la notion de `precondition` dans notre description des services permettra d'éviter la sélection des résultats des services « faux positifs » (se dit des services approuvés par l'algorithme de matching alors que réellement ils ne sont pas adaptés aux exigences spécifiques des clients de services Web).

Les postconditions

En programmation informatique, une postcondition est une condition (ou un prédicat) qui doit être toujours vraie, juste après l'exécution de certaines sections du code ou après une certaine opération dans une spécification formelle. En revenant à notre description de services, l'utilisation des postconditions permettra à l'utilisateur de définir ses conditions sur les résultats attendus de l'opération demandée.

Soit par exemple le service « e-bank », un service offrant à l'utilisateur un ensemble d'opérations telles que « consulter le compte », « calculer l'intérêt pour une certaine date à venir », « faire des transactions en ligne », etc. Notamment, un client cherchant à utiliser ce

type de service, peut exiger recevoir des notifications pour toute opération effectuée depuis son compte, sous forme de messages reçus sur son téléphone portable.

L'intégration de la notion de `postcondition` dans notre description des services permettra d'éviter la sélection des services Web qui ne répondent pas à des exigences spécifiques des clients, concernant les effets qu'ils attendent de l'exécution du service.

Ainsi, nous notons que les propriétés fonctionnelles pouvant être exprimées dans une requête seront représentées par les attributs «`Goal`, `Input`, `Output`, `Precondition et Postcondition` » relatifs à une `operation` d'un service WSDL.

5.3.3. Eléments d'extension non-fonctionnelle

Nous utilisons WS-Policy pour associer aux éléments d'un service WSDL ses `policies` décrivant les propriétés non-fonctionnelles exigées par le client. WS-Policy est une grammaire flexible et extensible qui permet d'exprimer les possibilités, les exigences, et les caractéristiques générales des services Web en tant que `policies`. Il a atteint le statut de recommandation W3C en 2007.

Dans le cadre de découverte de services Web, WS-Policy pourrait être utilisé pour permettre de définir les préférences des clients en matière de propriétés non-fonctionnelles, en particulier, en termes de qualité de service (QoS) tels que le niveau de sécurité, le temps de réponse, le prix du service et bien d'autres. C'est dans ce sens que nous l'intégrons dans la description des services Web au niveau de notre système. Ainsi, le client a la possibilité de spécifier des propriétés non-fonctionnelles qui seront représentées sous forme de `policies`. Comme exemples de propriétés non-fonctionnelles, nous citons les algorithmes de chiffrement supportés, le temps de réponse et le prix de l'opération à invoquer.

5.3.4. Notre modèle de services Web

L'extrait du diagramme de classes de la figure 5.5 donne une vue du modèle d'un service atomique manipulé dans notre système CBR4WSD. Ce diagramme reflète la description enrichie que nous avons proposée sur la base des standards préconisés par le W3C : WSDL 2.0 pour l'aspect fonctionnel, WS-Policy pour l'aspect non-fonctionnel et SAWSDL pour l'aspect sémantique [El Bitar et al., 2014a].

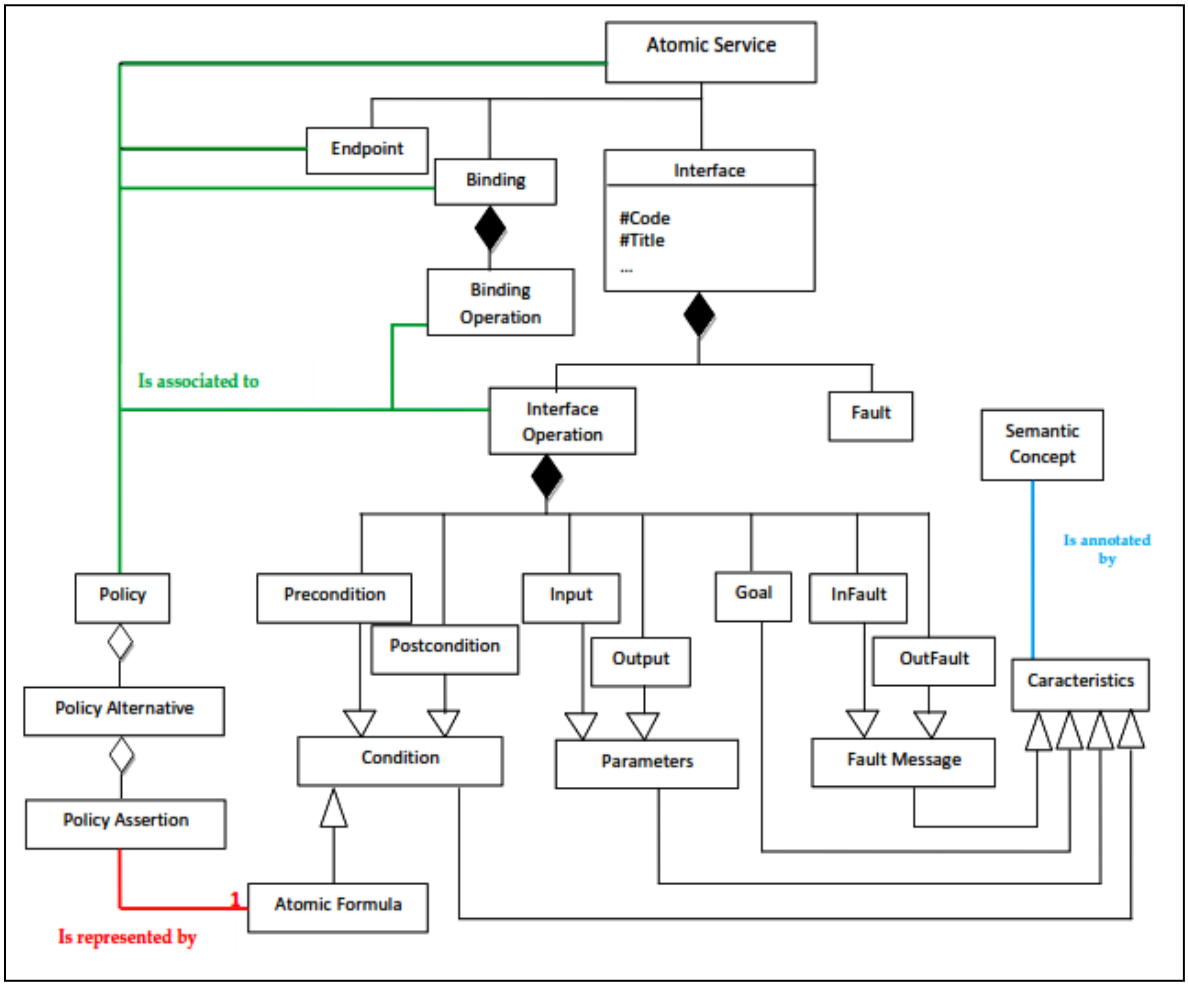


Figure 5.5: Extrait du diagramme de classes représentant un service atomique manipulé dans CBR4WSD.

Nous appelons « Atomic service » un service Web élémentaire. Tout service Web regroupe un ensemble d'opérations. Ces opérations sont décrites par les paramètres Input/Output et les Infault et Outfault auxquels nous avons rajoutés les éléments Goal, Precondition et Postcondition.

La classe « Caracteristic » généralise tous ces éléments. Elle est liée via l'association « is annotated » à un concept sémantique « Semantic Concept ». Cela signifie que les éléments peuvent être annotés conformément au standard d'annotation sémantique SAWSDL.

Par ailleurs, la classe « Condition » généralise les préconditions « Precondition », les Postconditions «Postcondition » et les formules atomiques « Atomic Formula ».

Finalement, conformément à WS-Policy, les aspects non-fonctionnels sont exprimés par la classe « `Policy` ». Une policy est décrite par un ensemble de policies alternatives « `Policy Alternative` ». Chaque « `Policy Alternative` » comprend un ensemble d'assertions décrites par la classe « `Policy Assertion` ». Nous notons qu'une « `Policy` » est associée au `service`, à l'`endpoint`, au `binding`, au `binding operation` et au `binding` paramètres (`input/output`) et `fault` tout en respectant les règles d'attachement d'une policy aux éléments de WSDL [WS-Policy W3C, 2007a] [WS-Policy W3C, 2007b].

Dans la suite, nous définissons les descripteurs du cas (*pb, sol(pb)*) utilisés dans notre système CBR4WSD et ce conformément aux standards adoptés pour la description des services Web.

5.4. Représentation de la partie problème d'un cas

La partie problème « *(pb)* » d'un cas reflète la requête du client cherchant une `operation` bien spécifique d'un service donné. La représentation de cette partie au niveau de notre système CBR4WSD, s'appuie sur notre définition du mécanisme de découverte de services Web comme étant "*l'acte de* ***localisation*** *d'une description, traitable par machine, d'un service Web non connu auparavant, décrivant certains* ***critères fonctionnels*** *et* ***non fonctionnels***".

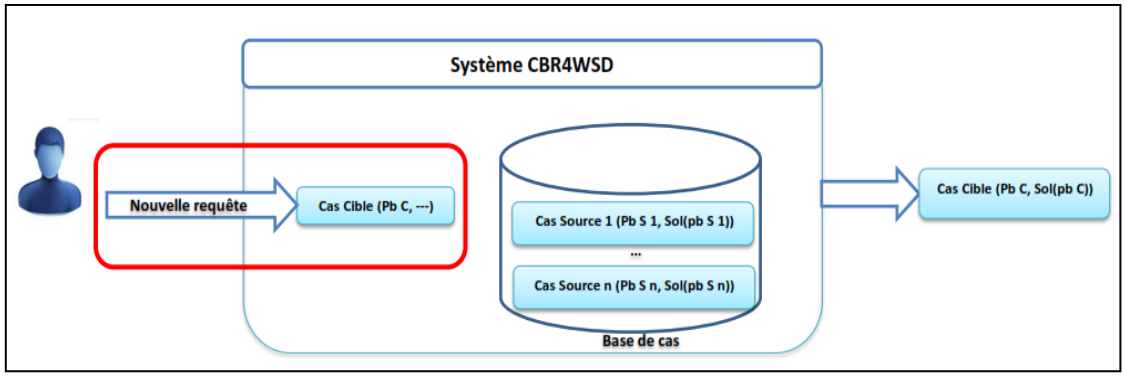


Figure 5.6 : Localisation du problème de représentation d'une requête par un cas cible dans le système CBR4WSD.

Ainsi, dans la partie problème, nous distinguons les propriétés fonctionnelles (PF) des propriétés non-fonctionnelles (PNF). Ces propriétés sont tirées de notre modèle enrichi de description de services Web. Notre cas se compose alors d'une partie fonctionnelle et d'une partie non-fonctionnelle dans son espace problème d'où la notation : *pb= (PF, PNF).*

5.4.1. Descripteurs fonctionnels de la partie problème d'un cas

Les propriétés fonctionnelles exprimées dans la partie « *(pb)* » du cas seront représentées par les attributs «`Goal`, `Input`, `Output`, `Precondition` et `Postconditon`» relatifs à une `operation` d'un service WSDL. Ainsi, les descripteurs de cette partie découlent de ces attributs comme illustré dans le tableau 5.2.

	Numéro et nom du descripteur	Définition
Descripteurs fonctionnels obligatoires	ds_1 : « Goal »	Objectif de l'opération du service SAWSDL requise.
	ds_2 : <Inputs>	Liste des paramètres d'entrée de l'opération requise.
	ds_3: <Outputs>	Liste des paramètres de sortie de l'opération requise.
Descripteurs fonctionnels optionnels	ds_4 : <Preconditions>	Liste des préconditions imposées sur l'opération requise.
	ds_5 : <Postconditions>	Liste des postconditions imposées sur l'opération requise.

Tableau 5.2: Descripteurs fonctionnels de la partie problème du Cas.

Dans l'étape d'élaboration du cas cible, les trois premiers descripteurs fonctionnels (ds_1, ds_2 et ds_3) sont absolument obligatoires et aucune recherche ne sera lancée sans que ces trois descripteurs ne soient renseignés.

Par contre, les descripteurs ds_4 et ds_5 sont optionnels et l'absence de leurs valeurs ne bloque pas la découverte mais elle peut conduire à des résultats « faux-positifs » surtout dans le cas de ds_4. Toutefois, l'existence d'information dans ces descripteurs leur donne automatiquement un aspect obligatoire à considérer lors de la découverte.

Spécification des descripteurs :

Formellement, les descripteurs de cette partie ne sont pas manipulés de la même manière du fait qu'ils appartiennent à deux catégories différentes :

1- les descripteurs de renseignement (ds_1, ds_2 et ds_3).

2- les descripteurs conditionnels (ds_4 et ds_5).

Notre système affecte à chacun des descripteurs un attribut relatif à la présence de l'information et qui est noté $\boldsymbol{ds_i^{Présence}}$. Dans les trois premiers descripteurs, cet attribut doit être égal à 1 et dans les descripteurs ds4 et ds5, il peut avoir la valeur 0 ou 1 selon la présence de l'information ou non.

En plus, en ce qui concerne les trois premiers descripteurs (descripteurs de renseignement), notre système affecte un deuxième attribut relatif à la valeur du descripteur noté $\boldsymbol{ds_i^{Valeur}}$. Celui-ci reflète le(s) concept(s) considéré(s) depuis le modèle hiérarchique de l'ontologie de domaine utilisée.

Les trois premiers descripteurs de la partie fonctionnelle d'un problème (ds_1, ds_2 et ds_3) ont par conséquent deux attributs relatifs à la présence de l'information dans le descripteur et la valeur du descripteur : $\boldsymbol{ds_i}$ = ($\boldsymbol{ds_i^{Présence}}$,$\boldsymbol{ds_i^{Valeur}}$) où :

- $\boldsymbol{ds_1}$ (goal),
- $\boldsymbol{ds_{2,j}}$ (input) avec j est le numéro de paramètre,
- $\boldsymbol{ds_{3,j}}$ (output) avec j est le numéro de paramètre.

Cependant, les descripteurs conditionnels ds4 et ds5 nécessitent une formalisation spéciale. Qu'il s'agisse d'une précondition ou bien d'une postcondition, le descripteur conditionnel exprime une condition qui doit être respectée au cours de la découverte.

Représentation d'une condition :

Le traitement automatique des conditions et des contraintes a connu depuis des années un grand succès autant du point de vue théorique et académique que du point de vue pratique et industriel. Ainsi, plusieurs travaux ont été présentés dans la littérature parmi lesquels nous citons : OCL (Object Constraint Language) [Warmer et Kleppe, 2003], LT (Logique Temporelle) [Benthem et Fak, 1977] et LPO (logique de Premier Ordre) [Codognet, 1995]. Ces travaux se distinguent par la puissance et l'efficacité de leur calcul dans divers domaines d'application, ceci par l'utilisation des outils efficaces dits résolveurs de contraintes.

Toutefois, l'utilisation de ces travaux nécessite une maîtrise du traitement des contraintes et des prédicats complexes, plus spécifiquement de l'écriture de programmes dans les langages déclaratifs ou spécifiques à la programmation de contraintes. Ceci constitue un obstacle majeur qui empêche l'utilisation de ces travaux pour l'expression des contraintes.

Ainsi, face à ce problème, nous avons choisi d'associer à une condition, quel que soit son type (précondition ou postcondition), une formule atomique énonçant une contrainte du client. Cette association nous facilite non seulement l'expression des conditions sous un format simple à manipuler par les utilisateurs de notre système, mais aussi le matching entre les descripteurs de la requête du client et leurs correspondants dans les services existants concernés par ces conditions.

Notre formule atomique représente une contrainte sur un concept donné de l'ontologie de domaine utilisée. Ce concept ontologique sera décrit par une valeur (instance) précise, et ceci via un opérateur qui peut être un opérateur de comparaison (=, !=, <, >, ≤ ou ≥) à titre d'exemple.

Composition de conditions fonctionnelles:

Nous rappelons que chacun des descripteurs ds_4 et ds_5 représente une liste qui peut contenir une ou plusieurs conditions élémentaires. Dans le contexte de contraintes complexes ou composites, nous distinguons l'utilisation des opérateurs « ET Logique » et « OU Logique ». Ce sont des fonctions logiques combinatoires, directement issues des mathématiques (algèbre de Boole), qui constituent les outils de base de la programmation de contraintes.

Lorsque les descripteurs ds_4 ou ds_5 comportent plusieurs conditions élémentaires, celles-ci sont implicitement liées entre elles par le « ET logique », formant ainsi une condition complexe exigeant la validité de toutes ses composantes.

En ce qui concerne le « OU Logique », dans le cas des préconditions et des postconditions relatives à un service Web, l'utilisation de cet opérateur n'a d'intérêt réel qu'au sein d'une même condition élémentaire. Ceci permet au fournisseur du service et au client d'exprimer une condition ouverte à plusieurs choix qu'ils pourront définir parmi les instances du concept en question. Nous illustrons l'utilisation des deux opérateurs logiques au sein du descripteur ds_4 par l'exemple suivant.

Soit un client cherchant un service de réservation de vols en ligne. Lors de la description de sa requête, ce client souhaite exprimer ses exigences en termes de préconditions par la liste suivante :

(PayementMode « Visa Electron » <u>OU</u> « American Express »)
<u>Et</u>
(Airline.Co « MEA Airlines »).

Il exprime deux préconditions. La première concerne le mode de paiement : le client exige que ce soit à travers une carte « Visa Electron » ou bien une carte « American Express ». Dans ce cas, l'opérateur « OU Logique » sera utilisé. La deuxième précondition concerne la compagnie aérienne. Le client exige exclusivement la réservation dans « Middle East Airlines ». Dans le traitement du descripteur ds_4, l'opérateur « Et Logique » sera considéré entre les deux préconditions, formant ainsi une condition complexe exigeant la validité de chacune de ses composantes.

Formalisation des descripteurs conditionnels ds_4 et ds_5 :

En revenant à la formalisation de nos conditions fonctionnelles, nous utilisons alors le 5-uplet suivant pour représenter une formule atomique telle que :

FA= (C, V, O, U, W) où :

- C : indique le concept en question. Normalement il doit être un concept de l'ontologie du domaine d'application (couleur de la voiture, etc.).
- V : représente l'instance ou la liste d'instances affectées au concept.
- O: indique un opérateur relationnel tels que (=, !=, <, >, ≤ ou ≥).
- U : représente l'unité si le concept est mesurable (grandeur quantitative).
- W : représente le poids (weight), par défaut W est égal à 1 dans ds_4 et ds_5.

Nous affectons six attributs à chacun des deux descripteurs conditionnels de la partie fonctionnelle du problème (ds_4 et ds_5) :

- $\boldsymbol{ds_i^{Présence}}$ **:** présence de l'information dans le descripteur,
- $\boldsymbol{ds_i^{Concept}}$ **:** concept en question,
- $\boldsymbol{ds_i^{Valeur}}$ **:** valeur du concept.
- $\boldsymbol{ds_i^{Opérateur}}$ **:** opérateur utilisé.
- $\boldsymbol{ds_i^{Unité}}$ **:** unité du concept.
- $\boldsymbol{ds_i^{weight}}$ **:** poids du descripteur (par défaut est égal à 1).

Ainsi les descripteurs ds_4 et ds_5 seront représentés sous la forme suivante :

$\boldsymbol{ds_{i,j}}$= ($\boldsymbol{ds_{i,j}^{Présence}}$, $\boldsymbol{ds_{i,j}^{Concept}}$, $\boldsymbol{ds_{i,j}^{Valeur}}$, $\boldsymbol{ds_{i,j}^{Opérateur}}$, $\boldsymbol{ds_{i,j}^{Unité}}$, $\boldsymbol{ds_{i,j}^{Weight}}$).

Le descripteur ds_6 (Communauté de services) :

Par ailleurs, le composant « élaborateur de cas cible » de notre système CBR4WSD est chargé de compléter la description du cas cible en l'annotant par la communauté de services à

laquelle il correspond. Il doit, à l'aide des descripteurs fonctionnels du problème cible, identifier à partir d'une base de communautés, celle à laquelle est associée le cas cible (cf chapitre 4, paragraphe 3.3).
Aussi, les cinq descripteurs présentés ci-dessus ne seront-ils pas les seuls considérés pour décrire fonctionnellement la partie problème du cas. Nous complétons cet ensemble de descripteurs fonctionnels par un descripteur clé noté ***ds6***. Ce descripteur original exprime une information décisive qui nous permet de sélectionner l'espace de recherche à considérer dans la phase de « Remémoration ». Formellement, il renseigne sur la communauté de services à laquelle correspond le cas de service.

Cependant, contrairement aux cinq premiers descripteurs fonctionnels (ds1 à ds5) dont les valeurs sont retrouvées directement dans la requête du client, la valeur affectée à ce descripteur clé sera déduite après avoir soumis la requête au système CBR4WSD. A l'aide du « Goal », descripteur fondamental de la partie fonctionnelle de la requête du client, nous affectons au descripteur ds6 l'identifiant de la communauté à laquelle est associé le cas cible.

Après avoir dégagé les descripteurs de la partie fonctionnelle du problème, nous présentons dans la suite les descripteurs de la partie non-fonctionnelle pour en ressortir ses descripteurs.

5.4.2. Descripteurs non-fonctionnels de la partie problème d'un cas

Nous rappelons que les propriétés non-fonctionnelles expriment des préférences ou conditions lors de l'interaction avec un service Web donné et qu'elles sont liées à différents domaines. A titre d'exemple, l'`endpoint` d'un service peut utiliser des messages signés par des algorithmes de chiffrement spécifiques. Cette propriété non-fonctionnelle précise le niveau de sécurité garanti par le service Web quand il est accessible par cet `endpoint`. Nous notons qu'un service Web utilisant différents `endpoints` peut fournir les mêmes propriétés fonctionnelles avec différents aspects non-fonctionnels. Ces aspects constituent, en effet, les critères essentiels du processus de sélection.

Nous exprimons la partie non-fonctionnelle du problème *(pb)* sous forme d'une politique alternative constituée d'une ou plusieurs formules atomiques. Chaque formule atomique représente une assertion qui correspond à une certaine préférence du client.

Par ailleurs, W3C définit quatre types d'éléments auxquels peuvent être associées des politiques : `Service`, `Endpoint`, `Operation` et `Message` [WS-Policy W3C, 2007a]. L'élément message fait référence à trois types de messages échangés: InMessage (message d'entrée), OutMessage (message de sortie) et FaultMessage (message d'erreur en entrée / sortie). Pour couvrir l'ensemble de ces éléments, nous définissons sept descripteurs non-fonctionnels dans la partie (pb) d'un cas, comme illustré dans le tableau 5.3.

	Numéro et nom du descripteur	**Définition**
Descripteurs non-fonctionnels (optionnels)	ds_7 : <ServicePolicy>	Liste des assertions relatives à l'élément service.
	ds_8 : <OperationPolicy>	Liste des assertions relatives à l'élément operation.
	ds_9 : <EndpointPolicy>	Liste des assertions relatives à l'élément endpoint.
	ds_{10} : <InMessagePolicy>	Liste des assertions relatives à l'élément InMessage.
	ds_{11} : <OutMessagePolicy>	Liste des assertions relatives à l'élément OutMessage.
	ds_{12} : <FaultMessagePolicy>	Liste des assertions relatives à l'élément FaultMessage.
	ds_{13} : <BindingPolicy>	Liste des assertions relatives au Binding.

Tableau 5.3: Descripteurs non fonctionnels de la partie problème du Cas.

Les descripteurs non-fonctionnels sont tous optionnels lors de l'élaboration du cas cible. L'absence de valeur à ce niveau ne bloque pas la découverte. Cette partie est spécifiquement utilisée pour permettre au client d'exprimer des préférences s'il le souhaite. Ce dernier peut spécifier plusieurs propriétés non-fonctionnelles. Toutefois, ces propriétés peuvent ne pas être prioritairement équitables pour lui. Il peut vouloir désigner des propriétés comme beaucoup plus importantes que d'autres, d'où la nécessité de lui permettre d'affecter un poids (weight) affecté à chaque propriété selon son importance.

Composition des conditions non-fonctionnelles :

Nous rappelons que chacun des descripteurs non-fonctionnels représente une liste qui peut contenir une ou plusieurs conditions élémentaires. Dans le cas où un descripteur non-fonctionnel comporte plusieurs conditions élémentaires (assertions), celles-ci sont implicitement liées entre elles par le « ET logique », formant ainsi une condition complexe

exigeant la validité de toutes ses composantes. En ce qui concerne l'emploi du « OU Logique », il permet d'exprimer plusieurs choix relatifs à une propriété non-fonctionnelle d'un service Web. Nous illustrons l'utilisation des deux opérateurs logiques en les appliquant au cas du descripteur ds8 (`Operation Policy`) dans l'exemple d'une requête qui cherche un service de réservation de vols en ligne. Cette requête doit être décrite comme suit :

(Language « English » OU « French »)

Et

(Security « RC4 »).

Elle exprime deux assertions. La première concerne le langage préféré, l'Anglais ou le Français. Dans cette situation, c'est l'opérateur « OU Logique » qui est utilisé. La deuxième assertion concerne la sécurité. Elle indique que c'est l'algorithme de chiffrement « RC4 » qui est préféré.

Formalisation d'une condition non-fonctionnelle :

Tout comme pour les préconditions et les postconditions, nous utilisons le 5-uplet (C, V, O, U, W) pour représenter une formule atomique décrivant une propriété non-fonctionnelle tels que :

- C : indique la caractéristique en question. Normalement, elle doit être un concept d'une ontologie spécifique de propriétés non-fonctionnelles (prix, temps de réponse, niveau de sécurité, etc.). Ceci n'empêche pas que ce paramètre soit un concept de l'ontologie du domaine d'application (couleur de la voiture, etc.).
- V : représente une valeur accordée au concept. Elle peut être une valeur quantitative ou qualitative (valeur numérique ou autres).
- U : représente l'unité associée au concept si jamais ce dernier est mesurable.
- O : indique un opérateur relationnel tels que (=, !=, <, >, ≤ ou ≥).
- W : représente le poids (weight) pour indiquer le degré d'importance d'une propriété non-fonctionnelle dans sa requête.

Par conséquent, chaque descripteur $\boldsymbol{ds_{i,j}}$ de la partie non-fonctionnelle aura aussi la forme suivante :

$\boldsymbol{ds_{i,j}}$= ($\boldsymbol{ds_{i,j}^{Présence}}$, $\boldsymbol{ds_{i,j}^{Concept}}$, $\boldsymbol{ds_{i,j}^{Valeur}}$, $\boldsymbol{ds_{i,j}^{Opérateur}}$, $\boldsymbol{ds_{i,j}^{Unité}}$, $\boldsymbol{ds_{i,j}^{Weight}}$).

Le tableau 5.4 illustre la structure globale de la partie problème d'un cas manipulé dans le système CBR4WSD:

Partie Problème													
Partie Fonctionnelle								**Partie non Fonctionnelle**					
ds_1	ds_2	ds_3	ds_4	ds_5	ds_6			ds_7	ds_8	ds_9	ds_{10}	ds_{11}	ds_{12}	ds_{13}
…	…	…	$ds^P_{4,1}$	$ds^P_{5,1}$	ds^P_6	ds^C_6	ds^V_6	$ds^P_{7,1}$	$ds^P_{8,1}$	$ds^P_{9,1}$	$ds^P_{10,1}$	$ds^P_{11,1}$	$ds^P_{12,1}$	$ds^P_{13,1}$
			$ds^C_{4,1}$	$ds^C_{5,1}$				$ds^C_{7,1}$	$ds^C_{8,1}$	$ds^C_{9,1}$	$ds^C_{10,1}$	$ds^C_{11,1}$	$ds^C_{12,1}$	$ds^C_{13,1}$
			$ds^O_{4,1}$	$ds^O_{5,1}$				$ds^V_{7,1}$	$ds^V_{8,1}$	$ds^V_{9,1}$	$ds^V_{10,1}$	$ds^V_{11,1}$	$ds^V_{12,1}$	$ds^V_{13,1}$
			$ds^V_{4,1}$	$ds^V_{5,1}$				$ds^O_{7,1}$	$ds^O_{8,1}$	$ds^O_{9,1}$	$ds^O_{10,1}$	$ds^O_{11,1}$	$ds^O_{12,1}$	$ds^O_{13,1}$
			$ds^U_{4,1}$	$ds^U_{5,1}$				$ds^U_{7,1}$	$ds^U_{8,1}$	$ds^U_{9,1}$	$ds^U_{10,1}$	$ds^U_{11,1}$	$ds^U_{12,1}$	$ds^U_{13,1}$
			$ds^W_{4,1}$	$ds^W_{5,1}$				$ds^W_{7,1}$	$ds^W_{8,1}$	$ds^W_{9,1}$	$ds^W_{10,1}$	$ds^W_{11,1}$	$ds^W_{12,1}$	$ds^W_{13,1}$
			…	…				…	…	…	…	…	…	…
			$ds^C_{4,n}$	$ds^C_{5,m}$				$ds^C_{7,p}$	$ds^C_{8,q}$	$ds^C_{9,r}$	$ds^C_{10,s}$	$ds^C_{11,t}$	$ds^C_{12,v}$	$ds^C_{13,x}$
			$ds^O_{4,n}$	$ds^O_{5,m}$				$ds^V_{7,p}$	$ds^V_{8,q}$	$ds^V_{9,r}$	$ds^V_{10,s}$	$ds^V_{11,t}$	$ds^V_{12,v}$	$ds^V_{13,x}$
			$ds^V_{4,n}$	$ds^V_{5,m}$				$ds^O_{7,p}$	$ds^O_{8,q}$	$ds^O_{9,r}$	$ds^O_{10,s}$	$ds^O_{11,t}$	$ds^O_{12,v}$	$ds^O_{13,x}$
			$ds^U_{4,n}$	$ds^U_{5,m}$				$ds^U_{7,p}$	$ds^U_{8,q}$	$ds^U_{9,r}$	$ds^U_{10,s}$	$ds^U_{11,t}$	$ds^U_{12,v}$	$ds^U_{13,x}$
			$ds^W_{4,n}$	$ds^W_{5,m}$				$ds^W_{7,p}$	$ds^W_{8,q}$	$ds^W_{9,r}$	$ds^W_{10,s}$	$ds^W_{11,t}$	$ds^W_{12,v}$	$ds^W_{13,x}$

Tableau 5.4 : Structure de la partie problème d'un cas du système CBR4WSD.

5.4.3. Transformation d'une description d'un service Web concret en un cas

Pour mieux comprendre cette migration du niveau description standard de service Web au niveau cas dans notre système CBR4WSD, nous avons choisi d'illustrer cette action par un exemple concret. Notre exemple traite un service Web chargé de la gestion de vols (réserver un vol, annuler une réservation, confirmer une réservation…).

Nous présentons dans la figure 5.7 un extrait de la partie fonctionnelle de notre service Web "Flight Manager" décrite en WSDL/SAWSDL et nous nous focalisons sur l'opération "bookFlight" (ligne 29) pour la réservation d'un vol. Cette opération est sémantiquement annotée conformément à notre modèle présenté dans la figure 5.4. L'attribut "`conceptType`" indique des référencements aux concepts ontologiques relatifs à l'opération "bookFlight" (cf. figure 5.7 - ligne 30). Cette opération est notamment annotée par les concepts suivants : "bookFlight" (`goal`), "paymentVisa" (`precondition`) et "sendEmailNotif" (`postcondition`).

En plus des trois nouveaux éléments que nous avons intégrés à la description WSDL/SAWSDL, il existe deux autres éléments descriptifs figurant dans l'opération "bookFlight". Il s'agit des éléments "FlightRequest" (`Input`) et "FlightResponse" (`Output`) qui, dans notre exemple, sont représentés par des structures complexes (lignes 4, 17) composées chacune d'un ensemble d'éléments simples.

```
1  <description>
2      <types>
3          <s:schema ...>
4              <s:element name="FlightRequest">
5                  <s:complexType>
6                      <s:sequence>
7                          <s:element minOccurs="0" maxOccurs="1" name="DepartureCity" type="s:string"/>
8                          <s:element minOccurs="0" maxOccurs="1" name="ArrivalCity" type="s:string"/>
9                          <s:element minOccurs="0" maxOccurs="1" name="DepartureDate" type="s:Date"/>
10                         <s:element minOccurs="0" maxOccurs="1" name="AirlineCo" type="s:string"/>
11                         <s:element minOccurs="0" maxOccurs="1" name="ClientName" type="s:string"/>
12                         <s:element minOccurs="0" maxOccurs="1" name="ClientFamily" type="s:string"/>
13                         <s:element minOccurs="0" maxOccurs="1" name="PassportNb" type="s:string"/>
14                     </s:sequence>
15                 </s:complexType>
16             </s:element>
17             <s:element name="FlightResponse">
18                 <s:complexType>
19                     <s:sequence>
20                         <s:element minOccurs="0" maxOccurs="1" name="TicketNb" type="s:string"/>
21                         <s:element minOccurs="0" maxOccurs="1" name="Price" type="s:Double"/>
22                     </s:sequence>
23                 </s:complexType>
24             </s:element>
25             ...
26         </s:schema>
27     </types>
28     <interface name="FlightManagerInterface" ...>
29         <operation name="bookFlight " pattern="http://www.w3.org/ns/wsdl/in-out "
30             conceptType="&ServiceOntology ; #goal &ServiceOntology ; #precondition &ServiceOntology ; #postcondition"
31             model Reference="&TravelOntology; #bookFlight &TravelOntology;#paymentVisa &TravelOntology;#sendEmailNotif" >
32                 <input  element ="FlightRequest "/ >
33                 <output  element ="FlightResponse"/ >
34         </operation>
```

Figure 5.7: Extrait de la description fonctionnelle d'un service de gestion de vols

Quant à la partie non-fonctionnelle de notre service, nous en présentons un extrait décrit selon WS-Policy et les formules atomiques proposées dans la figure 5.8. Ce service est décrit par deux propriétés non-fonctionnelles portant sur le temps de réponse "ResponseTime" (ligne 12) et le prix du service "ServicePrice" (ligne 18). Ces deux dernières assertions sont exprimées sous forme d'expression décrite par le 4-uplet (Concept, Value, Unit, Operator). Le temps de réponse doit être < 5 secondes et le prix égal à 0 USD.

```
1  <wsp:Policy
2      xmlns:wsp="..."
3      xmlns:wsse="..."
4      xmlns:qoso=".../ontology/operator"
5      xmlns:qosu=".../ontology/unit"
6      ...>
7
8      <wsp:ExactlyOne>
9          <wsp:All>
10         <wsp:Assertion name="RTassertion" ...>
11             <wsp:expression>
12                 <wsp:Concept> qosid:ResponseTime </wsp:Concept>
13                 <wsp:Value> 5 </wsp:Value>
14                 <wsp:Unit> qosu:seconds </wspes:Unit>
15                 <wsp:Operator> less </wsp:Operator>
16             </wsp:expression>
17             <wsp:expression>
18                 <wsp:Concept> qosid:ServicePrice</wsp:Concept>
19                 <wsp:Value> 0 </wsp:Value>
20                 <wsp:Unit> qosu:USD </wspes:Unit>
21                 <wsp:Operator> equal </wsp:Operator>
22             </wsp:expression>
23         </wsp:All>
24     </wsp:ExactlyOne>
25 </wsp:Policy>
```

Figure 5.8: Extrait de la description non-fonctionnelle associée à l'opération "bookFlight" du service de gestion de vols.

Exemple du cas représentant l'opération "bookFlight" :

Nous terminons notre exemple par la présentation du cas de service correspondant à la description de l'opération "bookFlight" que nous avons exposée.

Le tableau 5.5 représente la partie problème de notre cas qui reflète les détails des descriptions fonctionnelles et non-fonctionnelles de l'opération "bookFlight". Chaque partie possède ses propres descripteurs et à chaque descripteur correspond un ensemble d'attributs. Dans cet exemple de cas, nous utilisons sept descripteurs ($\boldsymbol{ds_1}$, $\boldsymbol{ds_2}$, $\boldsymbol{ds_3}$, $\boldsymbol{ds_4}$, $\boldsymbol{ds_5}$, $\boldsymbol{ds_6}$ et $\boldsymbol{ds_8}$) pour décrire la requête.

A noter que dans la partie non-fonctionnelle du cas figure un attribut poids affecté à chacune des assertions par le client. Cet attribut ne figure tout de même pas dans la description fournie par son fournisseur de service.

Partie Problème	Partie Fonctionnelle	ds_1	Goal	bookFlight	
		ds_2	Input	flightRequest	$ds^{C}_{2.1}$: DepartureCity
					$ds^{C}_{2.2}$: ArrivalCity
					$ds^{C}_{2.3}$: DepartureDate
					$ds^{C}_{2.4}$: AirlineCo
					$ds^{C}_{2.5}$: ClientName
					$ds^{C}_{2.6}$: ClientFamily
					$ds^{C}_{2.7}$: PassportNb
		ds_3	Output	flightResponse	$ds^{C}_{3.1}$: TicketNb
					$ds^{C}_{3.2}$: Price
		ds_4	Précondition	$ds^{C}_{4,1}$: PaymentMode	$ds^{O}_{4,1}$:=
				$ds^{V}_{4,1}$: paymentVisa	$ds^{U}_{4,1}$: ---
				$ds^{W}_{4,1}$: 1	
		ds_5	Postcondition	$ds^{C}_{5,1}$: Notification	$ds^{O}_{5,1}$:=
				$ds^{V}_{5,1}$: sendEmailNotif	$ds^{U}_{4,1}$: ---
				$ds^{W}_{5,1}$: 1	
		ds_6	ServCom	**Descripteur déduit (valeur accordée au cours du traitement)**	
	Partie non Fonctionnelle	ds_7	---	---	
		ds_8	Operation Policy	$ds^{C}_{8,1}$: ResponseTime	
				$ds^{O}_{8,1}$:<	
				$ds^{V}_{8,1}$: 5	
				$ds^{U}_{8,1}$: second	
				$ds^{W}_{8,1}$: 0,5	
				$ds^{C}_{8,2}$: ServicePrice	
				$ds^{O}_{8,2}$:=	
				$ds^{V}_{8,2}$: 0	
				$ds^{U}_{8,2}$: USD	
				$ds^{W}_{8,2}$:0,5	
		ds_9	---	---	
		ds_{10}	---	---	
		ds_{11}	---	---	
		ds_{12}	---	---	
		ds_{13}	---	---	

Tableau 5.5 : Partie problème du Cas représentant l'opération "bookFlight" du service "FlightManager".

5.5. Représentation de la partie solution d'un cas

Dans cette section, nous nous intéressons à la formalisation de la partie solution « *sol(pb)* » du cas. Nous détaillons dans la suite les types d'informations que nous pensons suffisantes à fournir au client souhaitant exploiter la solution obtenue.

5.5.1. Informations nécessaires pour l'exploitation de la solution

Pour invoquer une opération d'un service Web, le client doit avoir des renseignements qui lui permettent d'accéder et d'avoir des informations sur l'opération sollicitée. Parmi ces renseignements, figurent le service auquel appartient l'opération requise ainsi que les coordonnées du fournisseur de service auquel le client doit s'adresser en cas de problème.
Toutefois, ces informations seules ne suffisent pas pour permettre d'accéder à l'opération demandée, étant donné qu'un service qui implémente une multitude d'opérations, prévoit l'accès physique à ces opérations par différents `endpoints`. Chaque `endpoint` constitue un port d'accès physique au service. Il est donc impératif de connaître l'`endpoint` d'accès approprié et aussi le protocole de communication à utiliser. Par ailleurs, pour toute autre information sur les détails d'utilisation du service (par exemple, les schémas de données utilisés), le client peut se référer à son fichier WSDL.

Enfin, pour informer le client sur la pertinence de la solution et l'assister dans le choix de celle qui est pertinente parmi un ensemble d'éventuelles solutions, nous pensons qu'il serait opportun de l'informer des degrés de réponse de la solution à ses besoins fonctionnels d'une part, et non-fonctionnels d'autre part, mais aussi au niveau global.

Conformément aux besoins décrits ci-dessus, les informations liées à la partie solution du cas sont réparties suivant trois catégories:

1- Localisation du service et de l'opération. Il s'agit de présenter les informations concernant le fournisseur du service Web et l'URI de son site mais aussi les informations sur l'opération requise du service Web.
2- Propriétés techniques relatives à l'utilisateur de l'opération ciblée. Cette catégorie comprend les informations techniques nécessaires pour l'accès à l'opération demandée.
3- Degré de satisfaction du client. Cette catégorie est décrite par le pourcentage de satisfaction du client en termes de satisfaction des propriétés fonctionnelles, non fonctionnelles et aussi de satisfaction globale des deux types de propriétés demandées.

Nous utilisons dans la suite ces informations afin de modéliser la partie solution du cas.

5.5.2. Descripteurs de la partie solution du cas

Pour la description de la partie solution « *sol(pb)* », nous considérons six descripteurs comme montré dans le tableau 5.6.

	Numéro et nom du descripteur	Détails
Localisation	Ds_1 : Service	$Ds_{1,1}$: Nom du service
		$Ds_{1,2}$: URI du site du fournisseur
		$Ds_{1,3}$: Entreprise
		$Ds_{1,4}$: Coordonnées
		$Ds_{1,5}$: URI du fichier WSDL
	Ds_2 : Opération	$Ds_{2,1}$: Nom de l'opération
Propriétés techniques	Ds_3 : Accès	$Ds_{3,1}$: Protocole de communication
		$Ds_{3,2}$: Adresse d'accès
Satisfaction client	Ds_4 : Propriétés fonctionnelles	Ds_4 :% de satisfaction PF
	Ds_5 : Propriétés Non fonctionnelles	Ds_5 :% de satisfaction PNF
	Ds_6 : Globale	Ds_6 :%de satisfaction globale

Tableau 5.6: Descripteurs de la partie solution d'un cas du système CBR4WSD.

Enfin, pour illustrer la manière dont nous décrivons la partie solution « *sol(pb)* », nous reprenons la solutions de l'exemple « BookFlight » dont nous avons déjà présenté la partie problème « *(pb)* ». Le tableau 5.7 illustre le contenu de la partie solution de ce cas.

	Numéro et nom du descripteur	Détails	
Localisation	Ds_1 : Service	$Ds_{1,1}$: FlightManager	
		$Ds_{1,2}$: "http://emi.com/FlightManager"	
		$Ds_{1,3}$: EMI	
		$Ds_{1,4}$: Coordonnées	ElBitarCo.
			IbrahimElBitar
			00212622088448
			elbitar@emi.ac.ma
		$Ds_{1,5}$: "http://emi.com:8080/FlightManager?wsdl"	
	Ds_2 : Opération	$Ds_{2,1}$: flightBook	

Propriétés techniques	Ds_3 : Accès	$Ds_{3,1}$: http
		$Ds_{3,2}$: " http://emi.com:8080/FlightManager/flightBook
Satisfaction client	Ds_4 : Propriétés fonctionnelles	Ds_4 : 80%
	Ds_5 : Propriétés Nonfonctionnelles	Ds_5 : 80%
	Ds_6 : Globale	Ds_6 : 80%

Tableau 5.7: Partie solution du cas correspondant à "bookFlight".

Dans cette première partie du chapitre, nous avons explicité la manière dont nous formalisons un cas dans notre système CBR4WSD. Dans la suite nous abordons la phase d'élaboration, première phase du cycle RàPC, qui permet de retranscrire une requête de service en un cas cible.

5.6. Elaboration du cas cible

Selon Mille [Mille, 2006], la phase d'élaboration consiste à construire le problème cible à partir de l'entrée du système du RàPC. Cette phase comporte deux tâches essentielles : la création et la préparation du cas cible. Pour mieux comprendre, nous présentons ci-après, la manière dont nous avons exploité ces deux tâches dans le cadre de notre système CBR4WSD.

5.6.1. Création et préparation du cas cible

Lors de la phase d'élaboration, nous créons le nouveau cas conformément à la formalisation que nous avons proposée dans la première partie de ce chapitre. La création se fait automatiquement vu que la requête récupérée est décrite selon un Template conforme à cette formalisation. Par ailleurs, pour la tâche de préparation qui consiste à compléter, si possible, la description du problème en collectant ou déduisant d'autres informations pertinentes, le composant « élaborateur de cas cible » de notre système CBR4WSD est chargé de compléter la description du cas cible en l'annotant par la communauté de services à laquelle il correspond. Ce composant doit, à l'aide du descripteur fonctionnel « Goal » du problème cible, identifier à partir d'une base de communautés, celle à laquelle est associée le cas cible (cf chapitre 4, paragraphe 3.3). Autrement dit, il doit instancier le descripteur fonctionnel ***ds6*** qui renseigne sur la communauté de services à laquelle correspond le cas de service. Ce

descripteur original exprime une information décisive qui permet de sélectionner l'espace de recherche à considérer lors de la phase de « Remémoration ».
Par ailleurs, la tâche de préparation englobe également le choix des descripteurs jugés pertinents pour la résolution du problème et qui forment la base même de la recherche dans la base de cas. A ce niveau, nous avons déterminé les descripteurs indispensables au traitement de découverte. Nous avons notamment retenu des descripteurs obligatoires à savoir, « `Goal`, `Input` et `Output`» sans lesquels la découverte ne peut être lancée. Nous avons également considéré des poids à affecter aux autres descripteurs au choix de l'utilisateur.

Cependant, la phase d'élaboration n'est pas aussi simple. Elle pourra être accompagnée de problèmes liés aux connaissances manipulées dans les cas. Nous exploitons ces problèmes dans la section suivante.

5.6.2. Problèmes d'imperfection des connaissances

Le RàPC est un paradigme de résolution de problèmes qui s'appuie sur la réutilisation des expériences passées stockées dans une base de cas, pour résoudre de nouveaux problèmes appelés cas cibles. Généralement, qui dit expériences ou problèmes, dit cas concrets faisant intervenir des connaissances approximatives, mal définies ou vagues, et exprimées en langage humain. Ces imperfections de connaissances sont dues à des raisons liées à l'obtention de la connaissance par l'observation et à la représentation même de cette connaissance. Ils varient entre trois types différents [El Bitar, 2010] [El Bitar et al., 2011b]:

- L'incertitude : Elle traduit un doute sur la validité d'une connaissance. Ce problème provient de la difficulté d'obtention ou d'affirmation de la connaissance (par exemple, il serait difficile d'estimer le degré de la douleur (forte ou non) exprimée par un patient).
- L'imprécision : Elle concerne généralement les connaissances numériques. (par exemple, erreur de mesure de poids à 1% près).
- L'incomplétude : Elle reflète des absences de connaissances ou des connaissances partielles dans la description des cas. Elle est due à l'impossibilité d'obtenir certains renseignements ou bien à une négligence de certaines informations par l'utilisateur. (par exemple, les fichiers de malades dans lesquels certaines rubriques ne sont parfois pas remplies).

En fait, dans notre système CBR4WSD qui est spécifiquement conçu pour la découverte de services Web, nous ne sommes pas confrontés à tous les problèmes d'imperfection de connaissances cités auparavant. Les descripteurs du problème cible ne comportent pas des données pouvant supporter des incertitudes ou des imprécisions. Le problème cible manipulé ne reflète pas une perception comme dans le cas du diagnostic médical ou industriel où on peut facilement tomber sur des données imprécises et incertaines. Ceci est également le cas pour la solution cherchée. Ainsi, le seul problème d'imperfection de connaissances qui se pose dans notre cas, est celui de l'incomplétude du fait que la définition de nos descripteurs admette des descripteurs obligatoires et d'autres optionnels.

En effet, aucun cas cible ne sera généré si jamais le client essaie de lancer une requête avec des données incomplètes au niveau des descripteurs obligatoires (Goal, Inputs et outputs). Toutefois, si c'est le cas au niveau des données optionnelles, nous traitons le problème de l'incomplétude au cours de la phase de remémoration sans avoir à modifier ou remplir les descripteurs incomplets, en raison de l'impossibilité de prévoir ou d'imposer des préconditions ou des postconditions ou même des besoins en termes de propriétés non-fonctionnelles qui n'ont pas été fournies, au choix, par le client. Par contre, nous affectons tout simplement un poids nul à ces descripteurs.

5.7. Conclusion

Dans ce chapitre, nous avons focalisé sur la phase d'élaboration d'un cas dans notre système CBR4WSD dédié à la découverte automatique des services Web, et ce, en réponse à la requête d'un client demandant un service donné avec des besoins spécifiques.

Nous avons décrit la manière dont nous représentons un cas, tout en respectant notre objectif de départ, l'alignement avec les standards de services Web. Cette représentation est fondée sur un modèle de description de services Web sémantiques couvrant l'aspect fonctionnel aussi bien que l'aspect non-fonctionnel.

Outre la formalisation d'un cas, nous avons exposé les mécanismes adoptés pour sa préparation et qui consistent en la déduction du descripteur de communauté de services du cas et l'identification des descripteurs obligatoires ou importants pour la découverte.

Enfin, nous avons aussi exposé le problème d'incomplétude de connaissances que nous traitons par l'affectation de poids nul au descripteur de problème cible incomplet, et ce, au niveau de la phase de remémoration qui fera l'objet du chapitre suivant.

Chapitre 6

Organisation de la base de cas et remémoration

SOMMAIRE

6.1. Introduction

Notre système RàPC est construit autour d'une base de cas qui rassemble des enregistrements de cas de services appelés cas sources. Ces cas sont représentés conformément à la structure que nous avons décrite dans le chapitre 5. A la réception d'une requête, ils sont examinés en vue de retrouver les cas similaires au problème posé. Par la suite et conformément à l'objectif d'auto-apprentissage du RàPC, ils sont enrichis par le nouveau cas cible instancié selon le choix du client, selon les valeurs de la solution de l'un des cas sources similaires identifiés. Par conséquent, le nombre de cas sources est en mesure de croître de façon proportionnelle à l'utilisation du système. Toutefois, l'accroissement de la taille de la base de cas pose un véritable problème de performance lors de l'exploration des cas durant la phase de remémoration. En effet, bien que la capacité d'auto-apprentissage enrichisse la base de cas, la performance de résolution de problèmes est perturbée par le très grand nombre de cas qui devrait être considéré. Ceci se traduit alors par une baisse de la performance du système [Robeles, 2006].

Nous notons qu'un système de découverte de services Web ne serait intéressant qu'avec une base importante de cas de services. Cependant, nous pensons que la recherche de cas similaires devrait conserver une complexité constante au fur et à mesure que la base de cas s'enrichit. De ce fait, il serait opportun de penser à une méthode d'organisation de la base de cas, de façon à préserver la performance du système, en garantissant une recherche rapide des cas sources similaires au cas cible.

Dans cette même optique visant à rendre la phase de remémoration moins coûteuse en temps, nous suggérons de repenser la manière d'exécuter le processus de remémoration en proposant deux étapes : une étape offline qui s'exécute avant la réception de la requête et une étape online qui s'exécute de manière dynamique dès que la requête est reçue.

Outre le besoin d'assurer une remémoration accélérée, nous pensons qu'il est indispensable de définir un mécanisme de remémoration centré sur les besoins fonctionnels du client, assurant également une sélection efficace des services Web qui répondent à ses préférences non-fonctionnelles. Dans la littérature, peu de travaux prennent en considération les besoins non-fonctionnels ou les traitent au même niveau que ceux fonctionnels. Par ailleurs, nous pensons

que les services découverts doivent essentiellement répondre à la requête d'un client du point de vue fonctionnel, mais aussi satisfaire ses préférences non-fonctionnelles, particulièrement, en termes de qualité de service à titre d'exemple.

Dans la suite de ce chapitre, nous présentons notre approche d'organisation de la base de cas qui est un point focal dans le système CBR4WSD. Ensuite, nous exposons les détails de la phase de remémoration que nous réalisons en deux étapes : remémoration offline et remémoration online.

6.2. Base de cas et besoin d'organisation

L'organisation de la base de cas est dirigée par la nécessité de garantir une recherche rapide et efficace des cas sources les plus similaires au cas cible. Dans ce paragraphe, nous présentons les méthodes d'organisation de la base de cas et ensuite, nous mettons l'accent sur les éventuels critères d'organisation hiérarchisée pour la découverte de services Web.

6.2.1. Méthodes d'organisation de la base de cas

D'après Fuchs, il existe deux méthodes d'organisation de la base de cas : l'organisation simple et l'organisation hiérarchisée [Fuchs, 2008].

Dans l'organisation simple, les cas sont stockés en « vrac » sans aucune structuration. Durant la remémoration, lors de la recherche des cas similaires à un cas cible, il est obligatoire de scruter l'ensemble des cas de la base, afin d'en sélectionner le plus approprié. Toutefois, ce type d'organisation n'est pas recommandé lorsque le nombre de cas est considérable, en raison du processus d'appariement qui sera coûteux en termes de temps de recherche.

Prenons l'exemple d'une base de cas « en vrac » contenant n cas et une requête cherchant le cas x. Nous présentons dans le tableau 6.1 une comparaison des résultats obtenus en termes de temps de recherche pour n= 50 et n=100. Il est clair que dans ce type d'organisation, le temps de réponse est proportionnel au nombre de cas à scruter.

	Base de 50 cas en vrac	Base de 100 cas en vrac
Temps de recherche (1UT=temps de parcours d'un cas)	50 UT (Temps de parcours de 50 cas)	100 UT (temps de parcours de 100 cas)

Tableau 6.1 : Evaluation du temps de réponse pour une base de cas en vrac.

L'organisation hiérarchisée, quant à elle, est adoptée pour des bases de cas complexes. La base de cas est structurée de telle façon à permettre de rechercher un cas source de manière sélective, moyennant un processus de classification éliminatoire. Ceci permet de rétrécir l'espace de recherche à un sous-ensemble réduit, regroupant un ensemble restreint de cas à considérer. Soit l'exemple d'une base de cas hiérarchisée contenant n cas classés en huit classes et une requête cherchant le cas x. Nous considérons que le cas x recherché appartient à la classe 7 contenant 10 cas. Le tableau 6.2 présente une comparaison des résultats obtenus en termes de temps de recherche pour les cas où n= 50 et n=100.

	Base hiérarchisée de 50 cas	Base hiérarchisée de 100 cas
Temps de recherche	Temps de sélection de la classe 7 + Temps de parcours de 10 cas	Temps de sélection de la classe 7 + Temps de parcours de 10 cas

Tableau 6.2: Evaluation du temps de réponse dans une base de cas hiérarchisée.

Nous constatons que le temps de recherche est le même indépendamment de la taille de la base de cas. Il correspond aux temps de sélection de la classe correspondante au cas x et au temps de parcours des cas de la classe sélectionnée.

6.2.2. Critères d'organisation hiérarchisée pour la découverte des services

Dans le cas de la découverte de services Web, l'organisation hiérarchisée de la base de cas consiste à les organiser en groupements et à diriger la recherche directement vers le groupe de cas de services qui correspond au cas cible. Ceci permettra, sans doute, d'optimiser le processus de découverte en minimisant le temps d'appariement des cas par le fait de réduire l'espace de recherche et le limiter à un groupement de cas. Toutefois, la question qui se pose est sur la base de quel critère devra-t-on regrouper les cas de services au sein de la base de cas ?

En effet, le critère de regroupement dépend du domaine d'application. Dans le cas particulier des services Web, nous distinguons le regroupement par domaine métier [Ayorak et Bener, 2007] [Pilioura et Tsalgatidou, 2009] ou bien le regroupement par caractéristiques [Osman et al., 2006 a] [Osman et al., 2006 b] [Osman et al., 2009]. Ces types de regroupement permettent de regrouper les services Web se rapportant à un domaine particulier sans devoir considérer la spécificité de leurs fonctionnalités. Ainsi, des services ayant une même catégorie métier, mais assurant différentes fonctionnalités seront regroupés ensemble. Par exemple, dans le cas du domaine du e-tourisme, des cas correspondant à des services de fonctionnalités

différentes tels que la réservation de vols, la réservation d'hôtel et le e-paiement appartiendront à un même groupement.

D'après Sellami, le regroupement par domaine métier a marqué son efficacité dans le domaine de recherche d'information [Sellami, 2012]. Toutefois, nous trouvons que l'emploi de ce type de regroupement dans les registres de recherche de services Web est inadéquat, étant donné qu'il ne permet pas d'améliorer avec efficience le temps de réponse du processus de découverte. En effet, dans l'exemple d'un client cherchant un service lui permettant de regarder une course de chevaux en ligne, le processus de découverte pourrait être dirigé vers un groupement de services offrant entre autres des services de billetterie en ligne pour cette course.

Nous pensons que dans ce contexte, un client est à la recherche d'une opération bien spécifique d'un service Web assurant une fonctionnalité (un « goal ») bien déterminée. Ainsi, le regroupement des cas par domaine métier n'est pas adapté. Nous suggérons que le traitement de toute requête focalise plutôt sur la détermination de la fonctionnalité visée par cette requête et non pas la catégorie métier dont elle appartient. Dans ce sens, nous proposons un regroupement par besoins fonctionnels qui consiste à rassembler des cas répondant à un même ensemble de besoins fonctionnels, indépendamment de leurs propriétés non-fonctionnelles. Par exemple, tous les services assurant la fonctionnalité « réservation de vols en ligne » devraient être regroupés ensemble même si chacun d'eux est qualifié par un ensemble de propriétés non-fonctionnelles qui lui sont spécifiques (sécurité, temps de réponse, etc.). Aussi, organisons-nous la base de cas de notre système par besoins fonctionnels plutôt que par domaine métier. Les détails de la méthode d'organisation adoptée font l'objet de la section suivante.

6.3. Méthode d'organisation de la base de cas CBR4WSD

Dans un système de découverte de services Web à base de RàPC, le nombre important de cas de services que la base de cas peut comporter, complique sans doute le processus de découverte. C'est ainsi qu'une organisation non appropriée de la base de cas d'un système peut donner lieu à l'insatisfaction des clients due à des problèmes d'efficience, en particulier, de lenteur de temps de réponse. A cet effet, nous avons opté pour une organisation hiérarchisée de notre base de cas par « Communautés de services ». Cette organisation permet le

regroupement d'un nombre potentiellement grand de cas de services, indexés sémantiquement, par leurs besoins fonctionnels.

6.3.1. Communautés de services : définitions et objectifs

Le dictionnaire Oxford définit une communauté comme étant «un groupe de personnes vivant ensemble dans un même lieu et ayant certaines attitudes et certains intérêts en commun". Dans le domaine de services Web, Benatallah et al. définissent une communauté de services Web comme étant « *un ensemble de services Web ayant une fonctionnalité commune, mais des propriétés non-fonctionnelles bien différentes comme des fournisseurs et des paramètres de QoS différents* » [Benatallah et al., 2003]. Tout en respectant cette définition, Maamar et al. soulignent un autre usage possible du terme communauté, perçu comme « *un moyen pour fournir une description commune d'une fonctionnalité désirée sans se référer explicitement à un service Web concret qui mettra en œuvre cette fonctionnalité lors de l'exécution* » [Maamar et al., 2009].

Dans ce sens et en se référant à notre modèle de formalisation d'un cas de service, où la partie problème d'un cas de services reflète formellement une opération spécifique d'un service Web décrit par notre modèle de description sémantique de services Web (cf. chapitre 5), nous considérons la définition suivante : « Une communauté de services est l'ensemble de cas de services offrant des opérations de services Web qui assurent des fonctionnalités sémantiquement similaires mais des propriétés non-fonctionnelles pouvant être bien différentes. »

En effet, nous constatons que les communautés définies sont précisément des « communautés d'opérations de services ». Toutefois, nous maintenons le qualificatif « communautés de services » dans la suite de ce manuscrit, vu que les traitements des opérations de services sont généralement désignés dans la littérature comme des traitements de services et qu'une opération peut effectivement constituer un service. Nous utilisons par exemple l'expression de composition de services au lieu de composition d'opérations de services.

Par ailleurs, nous pensons que l'organisation de la base de cas par « communauté de services » est un moyen pour :

- faciliter d'abord la remémoration et la sélection des cas de services tout en contribuant à garantir un niveau satisfaisant de performance,
- fournir ensuite un cadre pour la substitution d'un service par un autre, sélectionné parmi ceux répondant le mieux à des besoins non-fonctionnels particuliers (par exemple, sur la base des critères de qualité de services).

Nous proposons dans la section suivante un modèle de communautés original que nous avons conçu pour représenter les communautés des cas de notre système CBR4WSD.

6.3.2. Notre modèle de communautés de services

En se référant à la requête qui représente une fonctionnalité demandée par le client, il serait possible de filtrer et réduire l'espace de recherche à la communauté de services offrant la fonctionnalité requise. Pour identifier les communautés du système CBR4WSD, nous proposons d'associer à chaque communauté une signature, que nous appelons « Web Service Community Signature » (WSCS). Cette signature reflète une fonctionnalité donnée et permet d'identifier les caractéristiques fonctionnelles des cas de la communauté concernée.

Selon notre modèle de description sémantique de services Web, les propriétés fonctionnelles exprimées dans la requête du client ou dans le cas cible, sont représentés par les attributs `Goal`, `Inputs`, `Outputs`, `Preconditions` et `Postconditons` relatifs à une `operation` d'un service (cf. chapitre 5, tableau 5.2). Dans cette section, nous montrons quels sont les attributs, parmi ceux-ci, qui constituent la signature WSCS de nos communautés.

Goal :

Le `goal` fait partie de l'ensemble des descripteurs fonctionnels obligatoires d'un cas dans notre système. Il identifie une fonctionnalité donnée. Nous notons que lors de l'élaboration du cas cible, le descripteur spécial (ds6) qui définit l'identifiant de la communauté correspondant au cas cible, est déduit depuis le `goal` (cf. chapitre 5, section 5.4.1). Une communauté regroupe notamment tous les cas ayant le même `goal`. Formellement, cet attribut est indispensable dans la signature d'une communauté du fait qu'il définit l'objectif fonctionnel du cas de service.

Preconditions et Postconditions :

Les `preconditions` et les `postconditions` sont des descripteurs fonctionnels optionnels pour le client. Il peut lancer sa requête sans devoir définir ces descripteurs. Ainsi, nous ne les avons pas retenus dans la définition d'une communauté.

Inputs et Outputs :

Une communauté peut regrouper des services ayant des descriptions ou caractéristiques différentes sans que ceci ne remette en question leur fonctionnalité commune. Des services peuvent notamment assurer le même `Goal` tout en ayant différents nombres d'`Inputs` et d'`Outputs`. Cette situation est normale étant donné le grand nombre de fournisseurs de services qui s'investissent pour concevoir et créer des services Web, chacun selon son point de vue et la quantité d'informations qu'il souhaite échanger avec son client. Nous considérons dans ce contexte l'exemple de deux services assurant la réservation de vol en ligne illustrés dans les figures 6.1 et 6.2.

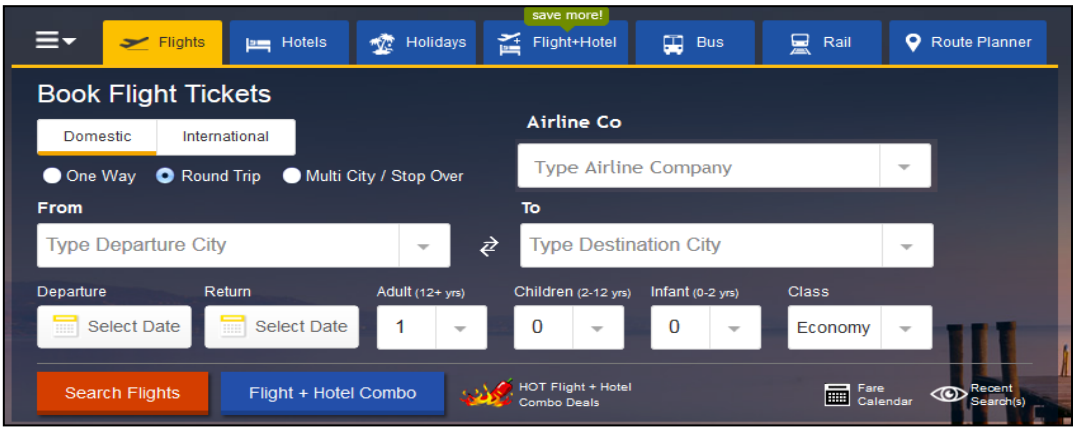


Figure 6.1 : Interface d'utilisation du service de réservation de vol en ligne « Book Flight Tickets »

Figure 6.2: Interface d'utilisation du service de réservation de vol en ligne « Flight Booking »

Les interfaces des deux services montrent les attributs Inputs nécessaires pour l'utilisation de chacun d'eux. Toutefois, nous remarquons que le service « Booking Flight Tickets » exige en plus du service « Fight Booking » deux paramètres d'entrée spécifiques :

- La compagnie aérienne (« Airline Company »),
- Et le type du vol « Domestic » ou « International ».

Dans ce cadre, si nous considérons que les services d'une communauté sont ceux qui ont nécessairement les mêmes `Inputs`/`Outputs`, de nombreux services qui assurent tous la même fonctionnalité ne seront pas regroupés ensemble. C'est exactement le cas des deux services pris comme exemples ci-dessus. Même s'ils assurent la même fonctionnalité, ils appartiendront à deux communautés différentes du simple fait qu'ils disposent d'un nombre d'`Inputs` différents. Cette manière de faire se contredit avec le principe de communauté de services.

Ainsi, nous avons pensé à une solution qui permet de considérer les `Inputs` et les `Outputs` dans la signature d'une communauté sans que leurs nombres ne soient fixes, autrement dit sans que les cas de cette communauté n'aient obligatoirement les mêmes `Inputs` et `Outputs`. En effet, nous représentons chaque `Input` ou `Output` d'un cas d'une communauté par un concept sémantique de l'ontologie métier correspondante. Tout concept sémantique annotant un `Input` ou `Output` d'un cas d'une communauté figurera dans sa signature. Ceci permettra d'identifier les différentes caractéristiques des services d'une communauté qui, notamment, regroupe des services ayant un même goal et différents concepts sémantiques comme paramètres `Inputs` et `Outputs`.
Par conséquent, nous définissons la signature d'une communauté de services par l'uplet (G, LC(I), LC(O)) où:

- G est le Goal assuré par les services de la communauté,
- LC(I) est la liste des concepts sémantiques annotant les `Inputs` des cas correspondant à des services de la communauté,
- LC(O) est la liste des concepts sémantiques annotant les `Outputs` des cas correspondant à des services de la communauté.

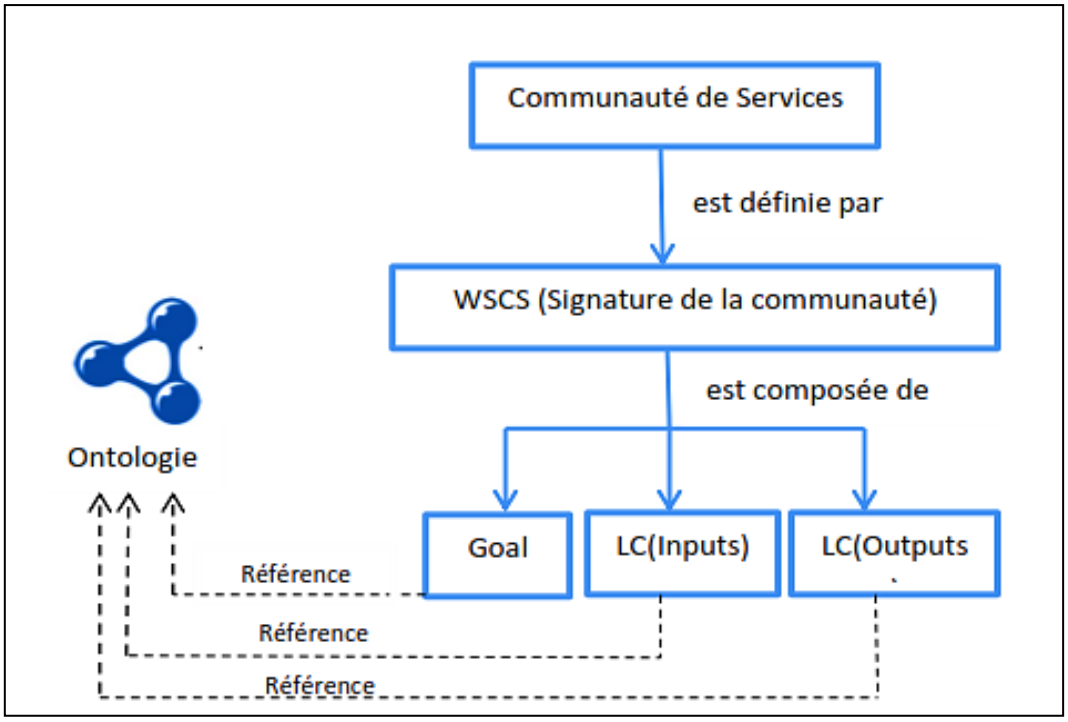


Figure 6.3: Notre modèle de communauté de services « WSCS ».

Les signatures des communautés de services manipulées dans notre base de cas sont sauvegardées dans une « Base de Communautés » qui sera utilisée lors du processus de remémoration.

6.3.3. Gestion des communautés de services

La méthode d'organisation de la base de cas que nous adoptons permet de procéder à la recherche d'un cas au moment de la remémoration, selon un processus de discrimination qui restreint l'ensemble des cas à considérer à une seule communauté (cf. figure 6.4).

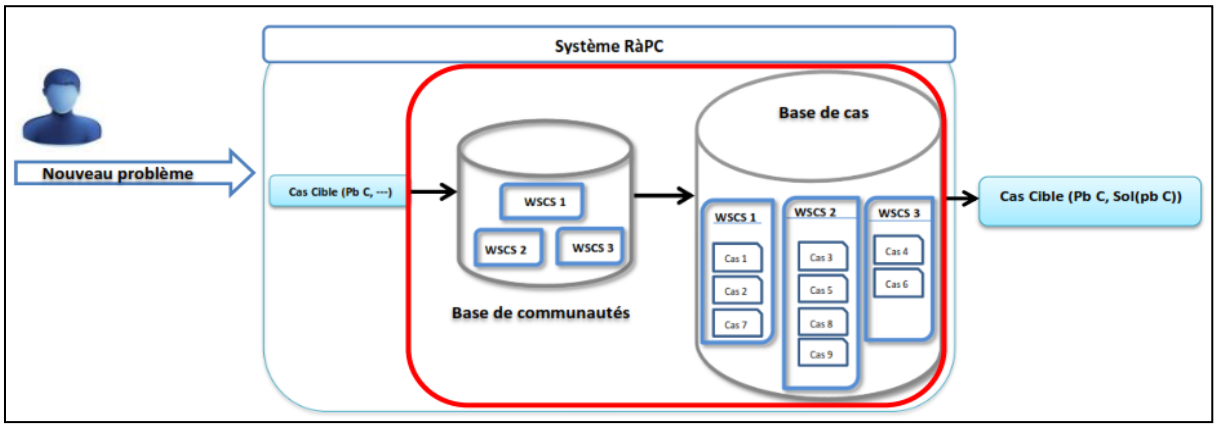


Figure 6.4: Localisation du problème de remémoration dans le système CBR4WSD.

Ce type d'organisation suscite la prise en considération d'opérations de gestion des communautés, qui seront déclenchées automatiquement ou suite à la demande de l'administrateur du système. Le but est de garantir la cohérence des communautés au cours de leurs cycles de vie. Nous avons ainsi identifié deux opérations : créer une communauté et adapter la signature d'une communauté qui doit être déclenchée suite à l'ajout ou la suppression d'un cas source.

Opération 1 : Créer des communautés

Conformément au processus offline de notre approche, après avoir alimenté la base de cas par un ensemble de cas sources, l'administrateur du système crée les communautés couvrant les concepts « Goal » de l'ontologie de domaine utilisée et les sauvegarde dans la base de communautés (cf. Chapitre 4 paragraphe 3.2). Chaque communauté possède une signature WSCS bien spécifique et regroupe l'ensemble des cas sources assurant le même goal que celui signalé dans sa signature. Toutefois, comme nous l'avons déjà évoqué, la signature n'est pas limitée à l'attribut Goal. Elle comporte aussi la liste des concepts sémantiques des Inputs et des Outputs LC(I) et LC(O).

Opération 2 : Adapter la signature d'une communauté

Les communautés de services seront gérées dans un environnement très dynamique où les changements sont fréquents. En effet, l'ajout d'un nouveaux cas dans la base de cas ou au contraire la suppression d'un cas entrainent la mise à jour de la communauté à laquelle ce cas se réfère. Cette mise à jour concerne particulièrement les listes LC(I) et LC(O) de la signature de la communauté concernée. Considérons la situation suivante :

- une communauté de services vide nommée (*Communauté1*) dont la signature est initialisée à (*G1*, <NULL>, <NULL>),
- deux cas *Cas1* et *Cas2* ayant respectivement les descriptions (*G1*, <I_n>, <O_n> …) et Cas2 (*G1*, <I_m>, <O_m> …).

Nous remarquons que ces cas se réfèrent au Goal *G1* et correspondent donc à des services de la communauté *Communauté1*. Leur ajout dans la base des cas doit être accompagné d'une adaptation de la signature de cette communauté au niveau des listes de concepts sémantiques LC(I) et LC(O). Aussi, la liste de concepts sémantiques Inputs de la communauté *Communauté1* doit-elle regrouper les concepts sémantiques Inputs de l'ensemble des cas

appartenant à cette même communauté (*Cas1* et *Cas2*). De même pour la liste des concepts sémantiques Output. Ainsi, les listes LC(I) et LC(O) de la *Communauté1* seront enrichies par les concepts sémantiques utilisés respectivement dans les inputs et les outputs de ses cas, à condition d'éviter la redondance des concepts dans chaque liste.

Nous illustrons dans la figure 6.5 le scénario d'adaptation de la signature de la communauté en fonction des nouveaux cas qui y sont introduits.

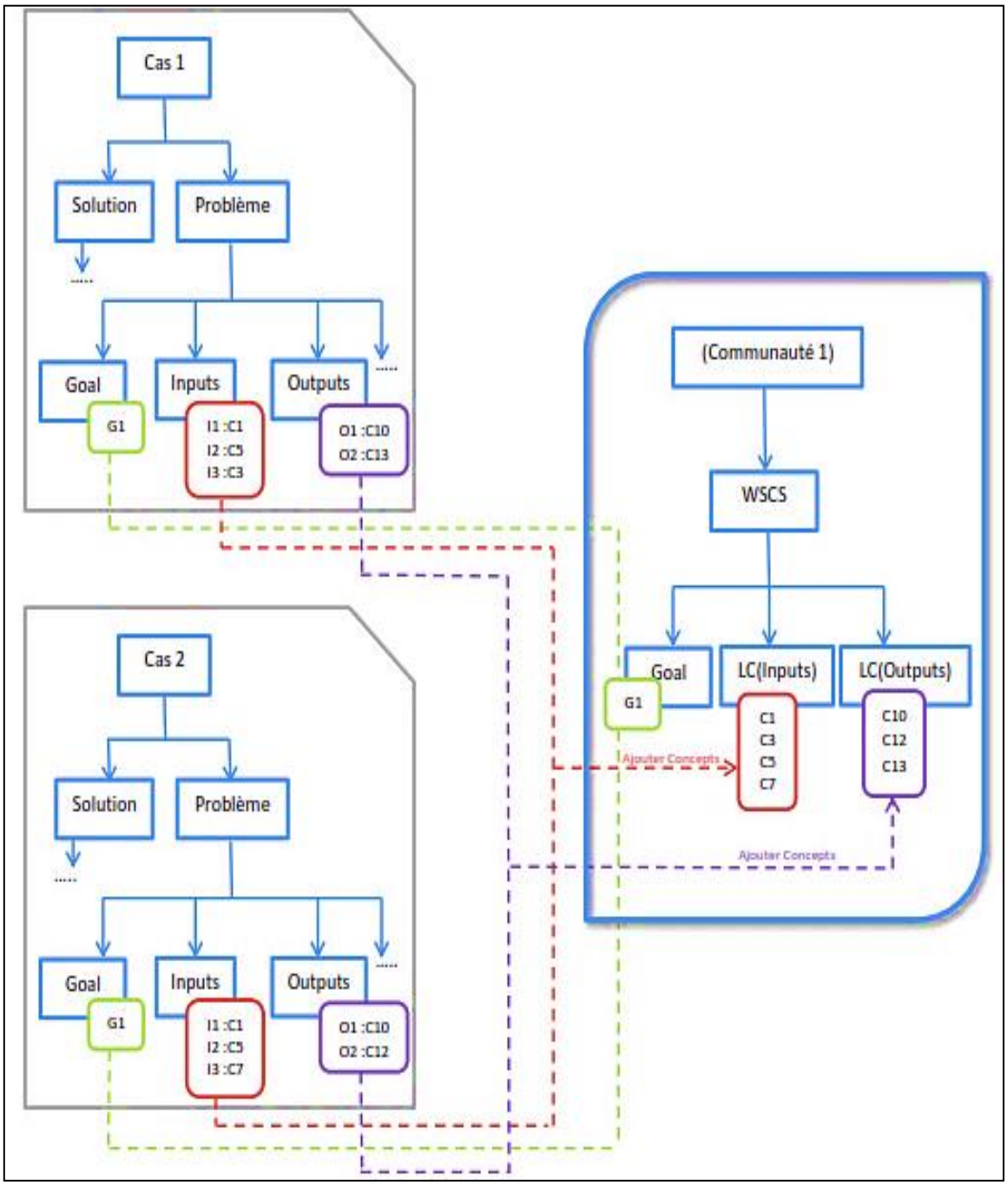


Figure 6.5: Scénario d'adaptation de la signature de la communauté

En ce qui concerne la suppression d'un cas depuis la base de cas, une mise à jour de la base de communautés, plus précisément la communauté de services à laquelle il réfère, sera également nécessaire. Ainsi, dans le cas de l'exemple que nous avons donné, le système doit supprimer les concepts sémantiques Inputs et Outputs associés au cas supprimé, des les listes LC(I) et LC(O) de la communauté *Communauté1* si jamais ces concepts ne sont pas utilisés par d'autres cas.

Finalement, nous notons que la mémorisation d'un nouveau cas résolu dans la base de cas ne devrait être autorisée que si son degré de similitude globale avec les cas sources n'est pas d'ordre Exact afin d'éviter des redondances au niveau de la base de cas. Par ailleurs, de nouveaux cas peuvent aussi être introduits directement par l'administrateur. Ce dernier, lorsqu'il reçoit la notification de l'échec de la remémoration correspondant à un cas cible donné, pourrait chercher une solution dans les registres UDDI et l'ajouter comme cas dans la base de cas du système. Nous notons aussi que la suppression d'un cas par l'administrateur peut avoir lieu si le système le lui notifie comme cas défaillant. En effet, si un cas est jugé défaillant par le système suite à des tests de vérification de l'état de son fonctionnement, l'administrateur sera notifié. Il pourrait par la suite supprimer le cas s'il s'assure que le service correspondant a été désactivé ou le mettre à jour s'il s'avère que l'URI du service a été changé.

6.3.4. Synthèse de l'organisation par communautés

Dans l'objectif de réduire le temps de recherche, nous avons opté pour une organisation de la base de cas par communautés de services. Le critère de regroupement est la fonctionnalité offerte par le cas de service (Goal) que nous considérons le premier critère de sélection. Chaque communauté est identifiée par sa signature composée de l'uplet (G, LC(I) et LC(O)). La gestion des communautés est assurée au moyen de deux opérations-clés : la création des communautés et la mise à jour de leurs signatures.

Cette organisation par communautés est mise à profit pour la recherche des cas dans la phase de remémoration. Rappelons que l'objectif de cette phase cruciale est de rechercher et sélectionner des cas de services sources relatifs à des requêtes similaires à la requête cible (cas cible). Elle consiste dans le cas de notre approche, à explorer la base de cas organisée par

communautés de services, pour découvrir ceux qui sont similaires au cas cible du point de vue fonctionnel et ensuite sélectionner les cas satisfaisant les propriétés non-fonctionnelles du client. Cela permettra d'optimiser le temps de découverte en ne cherchant que dans les cas associés à la communauté de service adéquate (cf. chapitre 6, section 6.2).

Nous montrons dans la suite comment s'opère cette recherche de cas, à travers les deux processus de remémoration offline et online.

6.4. Remémoration offline

Rappelons que la remémoration offline que nous proposons correspond à un travail préliminaire de calcul de similarité entre les concepts ontologiques mis en jeu. Elle est lancée par l'administrateur du système au cours du processus offline pour minimiser considérablement le temps réservé à l'appariement des cas sources avec le cas cible. L'objectif est double. D'une part, les calculs de similarité entre concepts qui sont coûteux en terme de temps sont réalisés à l'avance (mode offline). D'autre part, l'idée est d'éviter de dupliquer ce calcul de similarité entre un concept sémantique du cas cible et un autre qui figure dans plusieurs cas sources, en ne l'effectuant qu'une seule fois et en mémorisant ces résultats indépendamment des requêtes.

Le processus d'appariement sémantique doit, notamment, tenir compte des similitudes possibles entre les concepts annotant les différents descripteurs des cas appariés.
Par ailleurs, une ontologie permet de décrire sémantiquement les concepts d'un domaine donné ainsi que leurs différentes propriétés. Ces concepts sont reliés entre eux par des relations sémantiques donnant lieu à une structure hiérarchique d'ontologie. Les degrés de correspondance sémantique entre des concepts, appartenant à un arbre de l'ontologie (c'est-à-dire, liés par des relations sémantiques hiérarchiques par analogie à un arbre généalogique), sont généralement classés dans la littérature en quatre catégories [Hatzi et al. 2011] [Omrana, 2014], à savoir, les correspondances « Exact », « Plug-in », « Subsumes» et « Fail ».

Similitude « Exact »

La similitude entre deux concepts C1 et C2, notée *Sim(C1,C2),* est considérée comme une correspondance « Exact » si les deux concepts C1 et C2 sont équivalents, autrement dit si ces

concepts appartiennent à la même classe ontologique (annotés par la même URI dans une ontologie). Pour exprimer la correspondance « Exact » nous adoptons la notation : C1 ≡ C2.

Similitude « Plug-in »

La similitude entre deux concepts C1 et C2 est considérée comme une correspondance de type « Plug-in » si le concept C1 est une sous classe du concept C2, autrement dit, si l'URI utilisée pour annoter le concept C1, référence dans l'ontologie considérée, un concept défini comme une sous classe du concept référencé par l'URI utilisée pour annoter le concept C2. Pour exprimer la correspondance « Plug-in » des concepts C1 et C2, nous adoptons la notation : C1 ⊂ C2.

Similitude « Subsumes »

La similitude entre deux concepts C1 et C2 est considérée comme une correspondance de type « Subsumes » si le concept C1 est une superclasse du concept C2, autrement dit si l'URI utilisée pour annoter le concept C1, référence dans l'ontologie considérée, un concept défini comme une classe mère du concept référencé par l'URI utilisée pour annoter le concept C2. Pour exprimer la correspondance « Subsumes » des concepts C1 et C2, nous adoptons la notation : C1 ⊃ C2.

Similitude « Fail »

La similitude entre deux concepts C1 et C2 est considérée comme une correspondance de type « Fail » s'ils n'entretiennent aucune relation d'équivalence ou de parenté sémantique, autrement dit s'ils sont annotés par des URI qui référencent deux concepts ne possédant aucun lien d'équivalence ou de parenté sémantique dans l'ontologie où ils sont définis. Pour exprimer la correspondance « Fail » des concepts C1 et C2, nous adoptons la notation : C1 ≠ C2.

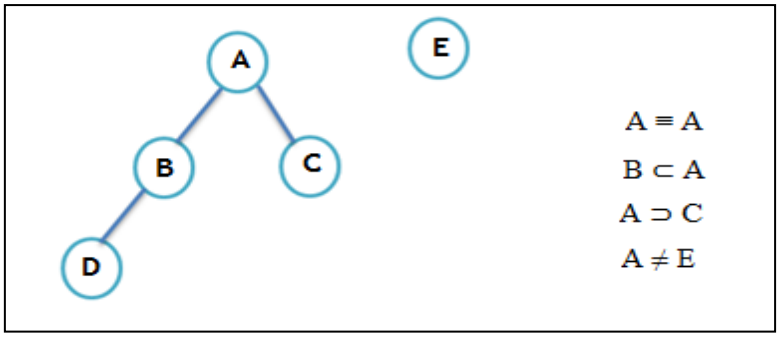


Figure 6.6: Exemple de résultats de matching sémantique dans une ontologie

A titre d'exemple, en appliquant ces règles de similitude inter-concepts aux concepts d'une ontologie de véhicules, nous obtenons les résultats rapportés dans la table de similitude illustrée par le tableau 6.3.

	Vehicule	Car	Motorcycle	Truck	FamilyCar	SportCar
Vehicule	**Exact**	**Subsumes**	**Subsumes**	**Subsumes**	**Fail**	**Fail**
Car	**Plugin**	**Exact**	**Fail**	**Fail**	**Subsumes**	**Subsumes**
Motorcycle	**Plugin**	**Fail**	**Exact**	**Fail**	**Fail**	**Fail**
Truck	**Plugin**	**Fail**	**Fail**	**Exact**	**Fail**	**Fail**
FamilyCar	**Fail**	**Plugin**	**Fail**	**Fail**	**Exact**	**Fail**
SportCar	**Fail**	**Plugin**	**Fail**	**Fail**	**Fail**	**Exact**

Tableau 6.3 : Extrait de la table de similitude correspondant aux concepts d'une ontologie de véhicules

Synthèse :

Le calcul de similarité entre les concepts utilisés pour décrire les cas consiste à identifier les relations reliant deux concepts : Exact, Plugin, Subsumes et Fail. Cela permet de créer une hiérarchie des concepts i.e. l'arbre d'ontologie. Ce calcul est effectué une seule fois puisqu'il est relatif aux concepts et non aux requêtes. Il sert à « alléger » le processus online de calcul de similarité entre deux cas, détaillé dans la section suivante.

6.5. Remémoration online

La remémoration online réutilise les résultats du processus offline de remémoration pour le calcul des similarités. Dans cette section, nous décrivons le processus global correspondant à cette phase, puis nous présentons ses fondements qui consistent en un ensemble de règles de vérification de découvrabilité et de sélection des candidats, en plus d'un mécanisme de calcul de similarités.

6.5.1. Processus global de remémoration online

Le processus de découverte démarre en fait, par la transformation des données de la requête récupérées à partir du « template » rempli par le client, en un cas cible aligné avec les standards W3C : WSDL 2.0, SAWSDL et WS-Policy 1.5. L'opération de génération du cas cible sera suivie de son élaboration dans le but de terminer l'étape de formalisation de ce cas (cf. chapitre 5). Ensuite, le processus de découverte se poursuit par le processus de remémoration online chargé de chercher des cas sources similaires au cas cible.

Sélection fonctionnelle :

A ce stade, le composant dit « Matchmaker sémantique online » applique des règles strictes de vérification de la découvrabilité du cas cible correspondant à la requête traitée. Si ce cas est découvrable, il applique ensuite la règle de sélection des cas sources candidats dont la communauté de services associée est équivalente à celle associée au cas cible. Après les avoir identifiés, il procède au calcul de leur similarité fonctionnelle (MSF) avec le cas cible. A l'achèvement de cette opération, un ensemble de cas sources dont la mesure de similarité calculée satisfait un seuil défini sera ensuite mis à la disposition du sélectionneur chargé d'en identifier le meilleur cas, relativement aux propriétés non-fonctionnelles demandées.

Sélection non-fonctionnelle :

Le composant sélectionneur calcule les mesures de similarité non-fonctionnelle (MSNF) des cas sources qu'il reçoit en entrée. Il est en mesure d'effectuer un « matching » appliqué aux politiques sources et cibles et de calculer par la suite, les mesures de similarité globale (MSG) entre les cas sources et le cas cible. Le but étant d'identifier les services Web à recommander au client. Ces services sont les solutions associées aux cas sources dont la mesure de similarité globale (MSG) est la plus élevée.

Incomplétude de connaissances :

Il est à noter que lors du calcul de similarités, aussi bien fonctionnelles que non-fonctionnelles, entre les cas sources et le cas cible, le traitement de l'incomplétude de connaissances, due à la non instanciation de certains descripteurs du problème est prévu dans notre approche. En effet, aucun cas cible ne sera généré si jamais le client essaie de lancer une requête avec des données incomplètes au niveau des descripteurs obligatoires (Goal, Inputs et outputs). Toutefois, si c'est le cas au niveau des descripteurs optionnels, le problème d'incomplétude sera tout simplement pris en considération lors du calcul de la similarité. En effet, un descripteur optionnel de type fonctionnel (précondition ou postcondition) qui n'a pas été spécifié par le client sera considéré comme non similaire au descripteur correspondant du cas source. Notamment, si le client ne spécifie pas une contrainte (précondition ou postcondition), c'est parce qu'il souhaite certainement ou très probablement utiliser le service dans un contexte qui ne la prend pas en considération. Par contre, un descripteur optionnel de type non-fonctionnel (propriété non-fonctionnelle) ne sera pas considéré lors du calcul de la

similarité. Si le client ne spécifie pas une propriété non-fonctionnelle, c'est parce qu'il souhaite certainement utiliser le service indépendamment de cette propriété.

6.5.2. Règles de vérification de la découvrabilité et de sélection des candidats

Les règles de vérification de la découvrabilité du nouveau cas cible que le processus de remémoration online applique à son déclenchement sont des règles strictes. Leurs résultats permettent d'informer le système de la possibilité de découvrir des services répondant à la requête soumise, à partir des cas sources de la base de cas. Nous notons que nous considérons qu'un cas cible n'est pas découvrable s'il est confirmé qu'il n'est pas possible de découvrir des services qui le résolvent à partir de la base de cas du système.

Dans ce contexte, nous utilisons deux règles de découvrabilité et nous considérons une troisième règle pour la sélection des cas sources candidats qui consiste à écarter ceux qui ne sont pas similaires :

Règle 1 : Vérifier que le descripteur de communauté de services du cas cible n'est pas nul.

Le composant élaborateur du cas cible du moteur RàPC du système, est chargé de compléter la description du cas cible en l'annotant par la communauté de services à laquelle il correspond. La phase de remémoration online consiste à vérifier d'abord si le descripteur correspondant n'est pas nul avant de lancer l'algorithme d'appariement.

Règle 2 : Vérifier que les descripteurs outputs du cas cible sont totalement couverts par sa communauté.

Dans le cas où la règle 1 est vérifiée, le processus de remémoration online poursuit le traitement de vérification de la découvrabilité du cas cible, en vérifiant une unique condition concernant ses outputs. En effet, la communauté identifiée du cas cible doit pouvoir satisfaire tous les outputs de la requête du client. Ainsi, chaque concept output du cas cible doit nécessairement avoir son concept correspondant de type « Exact » ou «Plugin » dans la liste des outputs de la communauté. Notons que :

- Une correspondance de type « Exact » indique que le concept output demandé figure dans la liste des outputs de la communauté. Ceci implique qu'un ou plusieurs cas sources de cette même communauté correspondent à des services qui fournissent cet output.
- Une correspondance de type « Plugin » indique que le concept père du concept output demandé figure dans la liste des outputs de la communauté. Ceci implique qu'un ou plusieurs cas sources de cette même communauté correspondent à des services qui fournissent ce concept père comme output. Nous citons par exemple le cas suivant :
    - Concept output demandé : Liste des voitures de marque « Renault ».
    - Concept output trouvé : Liste des véhicules de marque « Renault ».

Si au moins un output n'a pas son correspondant dans la liste des outputs de la communauté, le processus de découverte sera arrêté car il est certain que les services de la communauté concernée ne peuvent pas fournir toutes les sorties demandées par le client.

Par ailleurs, il n'existe aucune contrainte sur les inputs dans cette première étape. Les inputs du cas cible peuvent avoir des correspondants ou non dans la liste des inputs de la communauté. Ce qui importe à cette étape, ce sont les outputs (résultats attendus). Même si un input du cas cible ne figure pas parmi les éléments de la liste des inputs de la communauté, il ne serait pas possible de confirmer ou déduire qu'aucun cas source de la communauté concernée ne satisfait ce cas cible. Nous pouvons trouver un service assurant la fonctionnalité demandée qui n'impose pas d'introduire l'input en question. Dans un tel cas, si jamais ce cas source est choisi par le client, un nouveau cas sera inséré dans la base de cas et la liste d'inputs de sa communauté devra être mise à jour par l'insertion du nouveau concept input qu'il comporte.

Règle 3 : Règle de sélection des cas sources candidats

Lorsque les règles de découvrabilité sont vérifiées, le processus de remémoration online doit sélectionner les cas sources candidats pour lesquels il doit calculer la similarité avec le cas cible. Aussi, doit-il créer à partir des cas sources correspondant à la communauté identifiée du cas cible, l'ensemble des cas qui répondent fonctionnellement à ce cas cible. Ces cas doivent vérifier les conditions suivantes :

- Au niveau des outputs, le cas source doit fournir au minimum les outputs du cas cible. Nous ne prenons pas en considération un cas source qui ne satisfait pas la requête du client en termes d'outputs. Aussi, les outputs de la requête doivent-ils tous avoir leurs correspondants de type « Exact » ou « Plugin » dans les outputs d'un cas source pour qu'il puisse figurer dans la liste des cas découverts, sinon ce cas sera écarté.
- Au niveau des inputs, le cas ne doit pas exiger plus d'inputs que ceux spécifiés par le client. Il ne serait pas adéquat de proposer au client un service qui demande d'introduire des inputs que ce dernier n'est pas capable de fournir. Aussi, les inputs d'un cas source doivent-ils tous avoir leurs correspondants de type « Exact » ou « Subsumes » dans les inputs de la requête sinon ce cas sera écarté.

Après avoir identifié l'ensemble des cas sources vérifiant les besoins fonctionnels du client en termes d'outputs et d'inputs, le processus de remémoration online procède au calcul de la similarité fonctionnelle de chacun de ses cas sources avec le cas cible.

6.5.3. Mécanisme de calcul des similarités

Le mécanisme de calcul de similarité que nous avons adopté est organisé en trois différents niveaux : fonctionnel, non-fonctionnel et global. Rappelons que les similarités locales s'appliquent aux descripteurs de deux cas, alors que les similarités globales concernent deux cas.

6.5.3.1 Calcul de la similarité fonctionnelle

Pour calculer la similarité fonctionnelle entre un cas_i de l'ensemble E des cas sources candidats et le cas cible, nous procédons par agrégation des similarités locales effectuées sur chacun des descripteurs fonctionnels de la partie problème des deux cas appariés. Nous rappelons que cette partie est représentée par l'uplet de descripteurs ds_1 à ds_6 : (`Goal`, `Inputs`, `Outputs`, `Preconditions`, `Postconditions` , `ComId`).

Cas de ds1 (`Goal`) et ds6 identifiant de communauté (`ComId`) :

Le calcul de la similarité locale concernant les descripteurs ds_1 et ds_6 peut être tout simplement déduit du fait que les cas sources de l'ensemble E devant être appariés avec le cas cible, assurent le même Goal et appartiennent à la même communauté de services que ce dernier. Par contre, le calcul de la similarité locale entre des descripteurs de type ds_2 , ds_3 ,

ds_4 et ds_5 est complexe, vu que chacun de ces descripteurs est décrit par une liste d'attributs de taille variable et dont l'ordre des éléments n'est pas connu.

Cas de ds_2 (`Inputs`) et ds_3 (`outputs`) :

L'ensemble des cas sources candidats peut comprendre des cas qui correspondent à des services fournissant d'autres outputs en plus de ceux spécifiés par la requête du client. De même, la requête peut inclure d'autres inputs en plus de ceux exigés par des cas sources candidats. Nous notons que tous ces cas sources candidats restent susceptibles de répondre à la requête du client vu que les besoins fonctionnels en termes de goal, inputs et outputs se trouvent tous satisfaits. Aussi, le calcul de leur similarité avec le cas cible ne devrait-il pas poser de problèmes. Pour cela, nous limitons ce calcul aux outputs spécifiés au niveau de la requête et inputs exigés par un cas source.

Cas de ds_4 (`Preconditions`) et ds_5 (`Postconditions`) :

Le but est de chercher si les contraintes du client sont satisfaites par un cas source et de considérer que celles exigées uniquement par le cas source ne concordent probablement pas avec le contexte du client. Ainsi, nous calculons les similarités locales au niveau des descripteurs ds_4 ou ds_5 en se limitant aux préconditions ou postconditions spécifiées dans la requête. Les préconditions ou postconditions qui figurent uniquement au niveau du cas source auquel la requête est appariée ne seront ainsi pas considérées lors du calcul de similarité locale (leurs valeurs de similarité locale sont automatiquement nulles).

Toutefois, lors du calcul de leur similarité globale, il est indispensable de prendre en considération l'ensemble des préconditions ou de postconditions figurant aussi bien dans la requête que dans le cas source.

Calcul de similarité :

Le calcul des similarités locales est fondé dans notre contexte, sur le calcul de la distance sémantique entre des concepts dans une ontologie de domaine, tandis que la mesure de similarité fonctionnelle globale est calculée selon la mesure de Manhattan qui demeure largement adoptée dans le domaine des services Web. Nous notons que nous associons à tout descripteur fonctionnel un poids de valeur 1 car nous considérons que les descripteurs fonctionnels ont tous le même poids et sont tous primordiaux. Aussi, la formule que nous retenons pour le calcul de la similarité fonctionnelle entre les cas, est-elle la suivante:

Sim= $\frac{\sum_{i=1}^{n}[sim_i\,(a_i\ ,b_i\,)]}{n}$ où :

- n est le nombre d'attributs effectivement comparés parmi les attributs des différents descripteurs figurant dans les deux cas appariés,
- et sim_i la similarité locale calculée entre l'attribut i d'un cas source et un cas cible.

En fin du calcul des mesures de similarité fonctionnelle, le processus de remémoration online sauvegarde les résultats obtenus dans la colonne « SimF » d'une table de similarité **TabSim** que nous utilisons ensuite pour le choix des cas sources considérés comme satisfaisant les besoins fonctionnels du client. Le tableau 6.4 illustre un exemple du contenu de cette table obtenue à la fin de la phase de calcul des mesures de similarité fonctionnelle. Les cas dont la SimF est notée en vert ont un degré de similarité fonctionnelle supérieure ou égale au seuil de similarité fonctionnelle prédéfinie par l'administrateur (expert du domaine ciblé).

Cas sources	SimF	SimNF	SimG
cas_1	**0,86**	NULL	NULL
cas_2	**0,42**	NULL	NULL
cas_3	**0,66**	NULL	NULL
…	…	…	…
cas_n	**0,61**	NULL	NULL

Tableau 6.4 : Exemple illustrant l'état de la table TabSim en fin de calcul des mesures fonctionnelles

Nous notons que, seuls les cas ayant un degré de similarité supérieur au seuil de similarité fonctionnelle prédéfini (SeuilF) seront considérés à l'étape suivante de calcul des mesures de similarité non-fonctionnelle.

6.5.3.2 Calcul de similarité non-fonctionnelle

Dans le cadre du processus de remémoration online, le composant de sélection se charge de calculer les similarités non-fonctionnelles et globales en vue de sélectionner les services qui répondent au mieux à la requête du client. Nous notons que nous limitons le calcul de la similarité non-fonctionnelle aux cas sources candidats ayant un degré de similarité **SimF** supérieur ou égal au seuil prédéfini. Nous notons aussi que, en ce qui concerne les propriétés non-fonctionnelles, nous calculons la similarité à ce niveau en se limitant aux propriétés non-fonctionnelles spécifiées dans la requête. Autrement dit, le but est de chercher si les

préférences du client, auxquelles il associe un poids de valeur entre 0 et 1, sont satisfaites par un cas source apparié à la requête. Les propriétés non-fonctionnelles qui figurent uniquement au niveau de ce cas source ne seront ainsi pas considérées lors du calcul de la similarité locale ni lors du calcul de la similarité globale. Nous notons à ce stade que pour le calcul de la similarité non-fonctionnelle globale, nous associons à tout descripteur non-fonctionnel le poids défini par le client s'il est spécifié dans la requête et un poids nul sinon, et nous ne considérons que l'ensemble des propriétés non-fonctionnelles exprimées dans la requête. Aussi, la formule que nous adoptons pour le calcul de la mesure de similarité non-fonctionnelle globale, est celle de la distance pondérée de Manhattan :

$$\text{Sim}= \frac{\sum_{i=1}^{n} w_i \, [sim_i \, (a_i \, , b_i \,)]}{n \, \sum_{i=1}^{n} w_i}$$ où :

- n est le nombre d'attributs des différents descripteurs de la requête,
- w_i est le poids que le client affecte à l'attribut i selon son degré d'importance pour lui,
- et sim_i est la similarité locale calculée entre l'attribut i d'un descripteur au niveau du cas source et du cas cible.

En fin du calcul des mesures de similarité non-fonctionnelle, le processus de remémoration online (en particulier, le composant de sélection) sauvegarde les résultats obtenus dans la colonne « **SimNF** » de la table **TabSim** que nous utilisons ensuite pour la sélection des cas sources pertinents, devant satisfaire les besoins fonctionnels et non-fonctionnels du cas cible. Le tableau 6.5 illustre un exemple du contenu de cette table obtenu à la fin de la phase de calcul des mesures non-fonctionnelles. Les cas dont la **SimNF** est notée en vert ont un degré de similarité non-fonctionnelle supérieure ou égale au seuil de similarité non-fonctionnelle (SeuilNF) prédéfini par l'administrateur (expert du domaine ciblé).

Le cas de E	SimF	SimNF	SimG
cas_1	**0,86**	**0,66**	NULL
cas_2	**0,42**	NULL	NULL
cas_3	**0,66**	**0,70**	NULL
…	…	…	…
cas_n	**0,61**	NULL	NULL

Tableau 6.5 : Exemple illustrant l'état de la table TabSim en fin de calcul des mesures non-fonctionnelles

6.5.3.3 Calcul de similarité globale

Il est à noter que, lors de l'étape finale où la mesure de similarité globale (MSG) est calculée, seuls les cas ayant des degrés de similarité fonctionnelle et non-fonctionnelle supérieurs ou égaux respectivement au seuil de similarité fonctionnelle (SeuilF) et au seuil de similarité non-fonctionnelle (SeuilNF) seront considérés.

Nous notons aussi que le calcul des mesures de similarité globale consiste à calculer des moyennes des similarités fonctionnelles et non-fonctionnelles. A la fin de cette étape, le processus de remémoration online (en particulier, le composant de sélection) sauvegarde les résultats obtenus dans la colonne « **SimG** » de la table **TabSim**. Il sélectionnera les cas sources ayant une similarité globale supérieure ou égale au seuil de similarité globale (SeuilG) prédéfini par l'administrateur. Le tableau 6.6 illustre un exemple du contenu final de la table **TabSim**. Les cas dont la **SimG** est notée en vert ont un degré de similarité globale supérieure ou égale au seuil de similarité globale.

Le cas de E	SimF	SimNF	SimG
cas_1	**0,86**	**0,66**	**0,76**
cas_2	**0,42**	NULL	NULL
cas_3	**0,66**	**0,70**	**0,68**
…	…	…	…
cas_n	**0,61**	NULL	NULL

Tableau 6.6 : Exemple illustrant l'état final de la table TabSim

En fin de remémoration, le système retourne au client des fiches descriptives des solutions possibles. Chacune de ces fiches reflète la description de la partie solution « *sol(pb)* » d'un cas source pertinent. Les parties relatives aux descripteurs « Localisation » et « Propriétés Techniques » seront les mêmes dans le cas cible que dans le cas source similaire. Par contre, les parties relatives aux trois descripteurs Ds_4 *: Propriétés fonctionnelles,* Ds_5 *: Propriétés Non fonctionnelles* et Ds_6 *: Globale* (cf. chapitre 5, section 5.5.2) comprendront les valeurs obtenues lors du calcul des différentes mesures de similarité. Les valeurs de ces descripteurs ne peuvent être nécessairement les mêmes que celles du cas source qui sont liées à sa propre description « pb » et non pas à la description « pb » du cas cible.

6.5.3.4 Paramétrage du calcul de similarité

La procédure de calcul de similarité est paramétrée par trois seuils : seuil de similarité fonctionnelle (SeuilF), seuil de similarité non-fonctionnelle (SeuilNF) et seuil de similarité globale (SeuilG). A titre d'exemple, le seuil de similarité fonctionnelle décrit l'écart toléré entre les parties fonctionnelles des deux cas comparés.

En effet, le cas de similarité exacte (i.e. le cas source et le cas cible sont identiques) est rare. Aussi, introduisons-nous des règles plus flexibles de similarité entre les cas cible et source. Les seuils de similarité définissent cette tolérance. C'est l'administrateur de la base de cas qui les définit sur la base de son expertise. Nous pensons que la définition de ces critères est un travail de recherche à part entière.

6.6. Conclusion

Ce chapitre a été consacré à deux étapes importantes de notre travail : l'organisation de la base de cas et leur remémoration. Vu que notre système de découverte de services Web ne serait intéressant qu'avec une base importante de cas de services, nous avons proposé de l'organiser par communautés de services. Cette solution permet de conserver une complexité constante de la découverte au fur et à mesure que la base de cas s'enrichit. Nous pensons que notre proposition permettrait d'augmenter la performance du système CBR4WSD, tout en garantissant la recherche rapide des cas sources similaires au cas cible et la sélection des cas les plus adéquats.

En plus, il était indispensable dans la phase de remémoration, de définir un mécanisme centré sur les besoins particuliers de chaque client, contribuant à une sélection efficace des services Web les plus appropriés pour répondre à ces besoins. Pour ce faire, nous avons proposé un algorithme d'appariement sémantique qui commence par calculer la mesure de similarité fonctionnelle (MSF) consistant en l'agrégation des similarités locales effectuées sur les descripteurs fonctionnels des cas. Seuls les cas ayant un degré de similarité supérieur ou égal à un seuil de similarité fonctionnelle prédéfini par l'administrateur sont considérés à l'étape suivante où une mesure de similarité non-fonctionnelle est calculée. Cette mesure consiste en l'agrégation des similarités locales effectuées sur les descripteurs non-fonctionnels des cas.

Par la suite, seuls les cas ayant un degré de similarité supérieur ou égal à un seuil de similarité non-fonctionnelle prédéfini par l'administrateur sont considérés à l'étape finale où la mesure de similarité non-fonctionnelle est elle-même agrégée à la mesure de similarité fonctionnelle pour générer une mesure de similarité globale permettant d'identifier le ou les services les plus appropriés, capables de répondre au mieux aux besoins fonctionnels aussi bien que non-fonctionnels exprimés au niveau du cas cible.

L'implémentation de notre système fera l'objet du chapitre suivant.

Chapitre 7

Implémentation du système

SOMMAIRE

7.1. Introduction

Ce chapitre est consacré à la présentation des détails d'implémentation de notre approche. Nous y présentons l'architecture logicielle du système aussi bien que les principaux algorithmes traduisant les règles de découvrabilité et de sélection des candidats, et les fonctions de calcul de similarité relatives aux différentes parties descriptives d'un cas. Nous y exposons aussi l'interface homme machine que nous avons réalisée.

7.2. Architecture logicielle

L'architecture logicielle du CBR4WSD est organisée en 3-Tiers. Nous avons opté pour le pattern MVP (Model-View-Presenter) étant donné que ce pattern est convenable avec les interfaces riches (gestion de l'historique, défaire/refaire, etc.). Pour ce faire, nous avons utilisé l'outil GWT (Google Web Toolkit). La partie client (couche présentation) regroupe les composants Modèle, Vue et Presentateur du pattern MVP. La partie serveur est structurée en trois composants : Request Factory Servelet conformément au GWT, la couche métier qui représente la logique de traitement du système et la couche accès aux données. La troisième partie rassemble la base de cas et la base de communautés. La figure 7.1 illustre l'architecture applicative enrichie par nos choix technologiques.

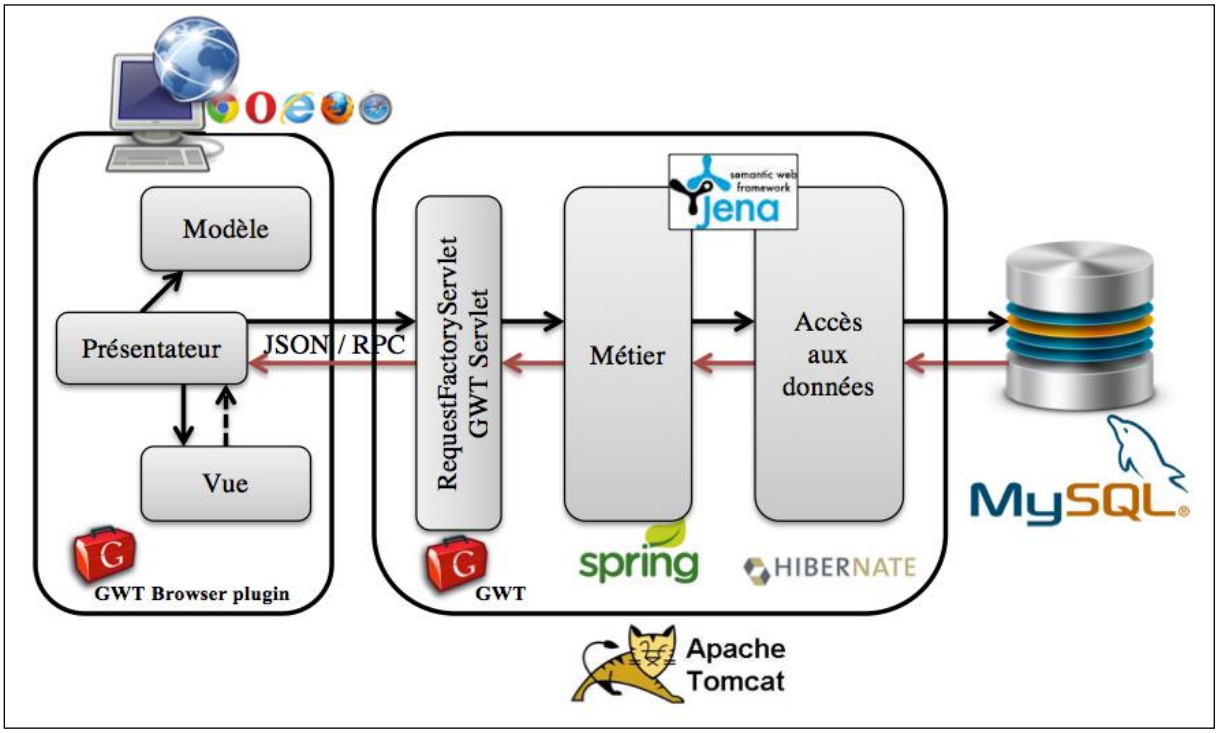


Figure 7.1 : Architecture logicielle du système CBR4WSD

Le choix en termes d'outils est dirigé par les exigences fonctionnelles de notre système. Une brève description de ces outils ou plateformes est présentée dans le tableau 7.1.

Outil / Plateforme	**Fonction**
Google Web Toolkit (GWT)	Créer et maintenir des applications web dynamiques
Spring	Construire et définir l'infrastructure d'une application java
Jena	Construction d'applications orientées Web sémantique
Hibernate	Gestion de la persistance des objets en base de données relationnelles
MySQL	Serveur de base de données
Jetty et Tomcat	Serveur Web
IDE Eclipse Platform : Juno	Environnement de développement libre

Tableau 7.1 : Outils et plateformes utilisés pour la mise en œuvre du système CBR4WSD

7.3. Implémentation du processus de remémoration

Pour l'implémentation du système CBR4WSD, nous avons élaboré un ensemble d'algorithmes relatifs aux processus de remémoration global qui englobe deux processus : offline et online.

Nous commençons par l'algorithme de remémoration offline en section 7.3.1.

Dans les sections suivantes (7.3.2 à 7.3.5), nous détaillons les différents algorithmes que nous avons développés pour mettre en œuvre le processus de remémoration online qui consiste en :

1- Un algorithme de vérification de la découvrabilité (7.3.2) qui cherche à :
   a. Vérifier que le descripteur de communauté de services du cas cible n'est pas nul,
   b. Vérifier que les outputs du cas cible sont totalement couverts par sa communauté.

2- Un algorithme de sélection des cas sources candidats depuis la communauté choisie (7.3.3) qui cherche à :

a. Vérifier que tous les outputs du cas cible ont leurs correspondants de type Exact ou Plugin dans la liste des outputs du cas source.
b. Vérifier que tous les inputs du cas source ont leurs correspondants de type Exact ou Subsumes dans la liste des outputs du cas cible.

3- Algorithme de calcul de similarité fonctionnelle entre le cas cible et les cas sources sélectionnés (7.3.4) qui cherche à :
   a. Calculer les similarités locales au niveau des descripteurs fonctionnels des deux cas comparés.
   b. Générer la similarité fonctionnelle globale entre les deux cas comparés.

4- Algorithme de calcul de similarité non-fonctionnelle entre le cas cible et les cas sources sélectionnés (7.3.5) qui cherche à :
   a. Sélectionner les cas sources ayant un degré de similarité fonctionnelle supérieure au seuil de similarité fonctionnelle SeuilF.
   b. Calculer les similarités locales au niveau des descripteurs non-fonctionnels des deux cas comparés.
   c. Générer la similarité non-fonctionnelle globale entre les deux cas comparés.

5- Algorithme de calcul de similarité globale entre le cas cible et les cas sources sélectionnés (7.3.5) qui cherche à :
   a. Sélectionner les cas sources ayant un degré de similarité non-fonctionnelle supérieure au seuil de similarité non-fonctionnelle SeuilNF.
   b. Calculer la similarité globale entre des deux cas comparés.

7.3.1. Algorithme de calcul de la similarité offline

En ce qui concerne le processus de remémoration offline, il s'agit des algorithmes de gestion de la base de cas et de la base de communautés de services, d'un algorithme de configuration des seuils de similarité aussi bien qu'un algorithme d'appariement sémantique que nous avons implémenté en réutilisant l'API Jena [Jena, 2014]. Celui-ci consiste à créer la matrice de similarité TabSimFonc (cf. figure 7.2).

```java
/*
*** fonction d'appariment sémantique offline et
 de création de la matrice de similitude inter-concepts
*/
public void creerMatrice(Ontology Ont) {
    int taille = Ont.gerConceptNb();
    String [][] TabSimFonc = new String[taille][taille];
    Iterator<String> itConcepts = Ont.iterator();

    //Remplissage des entetes par les concepts
    for(int i=1; i<taille;i++)
    {
        if (itConcepts.hasNext())
        {
            String concept = itConcepts.next().toString();
            TabSimFonc[0][i] = concept;
            TabSimFonc[i][0] = concept;
            TabSimFonc[i][i] = "Exact";
        }
    }
    for (int i = 1; i < taille; i++)
    {
        for (int j = 1; j < taille; j++)
        {
            if (i != j)
            {
                String relation = getRelation(TabSimFonc[i][0], TabSimFonc[0][j]);
                if (relation.equals("subClassOf"))
                {
                    TabSimFonc[i][j] = "Subsumes";
                    TabSimFonc[j][i] = "Plugin";
                    continue;
                }
                if (relation.equals("Fail") && matrice[i][j] == null && matrice[j][i] == null)
                {
                    matrice[i][j] = "Fail";
                    matrice[j][i] = "Fail";
                }
            }
        }
    }
}
```

Figure 7.2 : Algorithme de similarité offline (inter-concepts ontologiques)

7.3.2. Algorithme de vérification de la découvrabilité

Le processus de remémoration online démarre par la vérification des règles de découvrabilité du nouveau cas cible. Pour ce faire, nous considérons deux fonctions : la fonction *Verify_Community* et la fonction *VerifyOutpTotalMatch*.

La fonction *Verify_Community* prend le cas cible élaboré comme paramètre d'entrée et retourne la valeur « true » si le descripteur de la communauté correspondante n'est pas nul. Sinon, une notification est renvoyée à l'administrateur et la procédure de découverte se

termine en retournant la valeur « false », après avoir affiché un message d'échec au client. Le détail de cet algorithme constituant cette fonction est donnée dans la figure 7.3.

```
boolean Verify_Community (Target_Case Req)
{
    <mysql> $query = $this.query(SELECT COUNT(*) FROM Community
        ->    WHERE Community.Id=Req.ComId);
        -> if ($query.num_rows() == 0)  // Le résultat "0" indique que la communauté est vide
            {

                System.println ("aucun cas correspondant existe\n") ;
                Notify_Admin(); //Transférer la requête a l'administrateur du système
                return (False); //Arreter le traitement
            }
          else return (True); //Poursuivre le traitement de vérification avec la règle 2
}
```

Figure 7.3 : Algorithme de vérification du descripteur de communauté du cas cible

Quant à la fonction *VerifyOutpTotalMatch*, elle vérifie si tous les outputs de la requête ont leurs correspondants de type Exact ou Plugin dans la liste des outputs de la communauté identifiée pour la requête en question. Cette fonction prend comme paramètres d'entrée le cas cible élaboré « *TargetCase Req* » et la communauté qui lui est associée « *Community Com* » et retourne la valeur « true » si tous les outputs de la requête ont leurs correspondants de type Exact ou Plugin dans la liste des outputs de la communauté (cf. figure 7.4).

```
/*
***fonction qui vérifie si tous les Outputs de la requete***
***ont leurs correspondants dans la liste des outputs de la communauté***
*/
boolean VerifyOutpTotalMatch (TargetCase Req, Community Com)
{
    LoutpR= Req.getListeOutput() ;
    nbOutp = LOutpR.getnb();
    LoutpCom= Com.getListeOutput() ;

    int i=0;
    boolean response =true;
    while (i< nbOutp)
    {
        if haveOutputMatch (LoutpR[i], LOutpCom)
            {
                i ++;
            }
        else
            {
                response =false;
            }
    }
    return (response);
```

Figure 7.4 : Algorithme de la fonction VerifyOutpTotalMatch

Comme nous le constatons, la fonction *VerifyOutpTotalMatch* fait appel à une autre fonction appelée *haveOutputMatch* qui vérifie si un output de la requête a son correspondant de type Exact ou plugin dans la liste des outputs de la communauté. Pour ce faire, elle prend comme paramètres d'entrée un *output o* de la requête et la liste des outputs de la communauté *LOutpCom*. Elle retourne la valeur « true » si l'output a son correspondant de type Exact ou Plugin dans la liste *LOutpCom* (cf. figure 7.5).

```
/*
***fonction qui vérifie si un output de la requête****
***a son correspondant dans la liste des outputs de la communauté***
*/
boolean haveOutputMatch(output o, listConcepts LOutpCom)
{
     nbOutp = LOutpCom.getnb();
     int j=0;
     boolean response =False;

     while (j< nbOutp)
          {
               if ((TabSimFonc (o, LoutpCom[i]) = = "Exact") ||
                (TabSimFonc (o, LoutpCom[i]) = = "Plugin")
                    {
                         response=true;
                         break;
                    }
               else
                    {
                         j++;
                    }
          }
     return (response);
}
```

Figure 7.5: Algorithme de la fonction HaveOutputMatch

7.3.3. Algorithme de sélection des cas sources candidats

Pour la sélection des cas sources candidats, nous considérons la fonction *Create_E_Set* qui fait appel à deux fonctions *VerifyCaseOutpTotalMatch* et *VerifyCaseInpTotalMatch*. La fonction *VerifyCaseOutpTotalMatch* prend comme paramètres d'entrée un cas source « *SourceCase Cas* » et le cas cible élaboré « *TargetCase Req* » et retourne la valeur « true » si tous les outputs de la requête ont leurs correspondants de type « Exact » ou « Plugin » dans la liste des outputs du cas source (cf. figure 7.6).

```
/*
***fonction qui vérifie si tous les Outputs du cas source***
***ont leurs correspondants dans la liste des outputs de la requete ***
*/
boolean VerifyCaseOutpTotalMatch (SourceCase Cas, TargetCase Req)
{
     LoutpC= Cas.getListeOutput() ;
     LoutpR= Req.getListeOutput() ;
     nbOutp = Req.getnb();

     int i=0;
     boolean response =true;
     while (i< nbOutp)
     {
          if haveCaseOutputMatch (LoutpR[i], LOutpC)
          {
               i ++;
          }
          else
          {
               response =false;
               break;
          }
     }
     return (response);
}
```

Figure 7.6: Algorithme de la fonction VerifyCaseOutpTotalMatch

Cette fonction fait appel à une autre fonction appelée haveCaseOutputMatch qui vérifie si un output de la requête possède son correspondant de type « Exact » ou « Plugin » dans la liste des outputs d'un cas source. Pour ce faire, elle prend comme paramètres d'entrée un output o de la requête et la liste des outputs du cas LOutpC. Elle retourne la valeur « true» si elle trouve un correspondant de l'output o de type « Exact » ou « Plugin » dans la liste LOutpC. Sinon, elle retourne la valeur « false » (cf. figure 7.7).

```
/*
***fonction qui vérifie si un output de la requete***
***a son correspondant dans la liste des Outputs du cas source***
*/
boolean haveCaseOutputMatch(output o, list LOutpC)
{
    nbOutp = LOutpC.getnb();
    int j=0;
    bool response =false;

    while (j< nbOutp)
    {
     if ((TabSimFonc (o, LoutpC[j]) = = "Exact") ||
     (TabSimFonc (o, LoutpC[j]) = = "Plugin")
        {
            response=true;
            break;
        }
        else
        {
            j++;
        }
    }
    return (response);
}
```

Figure 7.7: Algorithme de la fonction haveCaseOutputMatch

Par ailleurs, la fonction *VerifyCaseInpTotalMatch* prend comme paramètres d'entrée un cas source « *SourceCase Cas* » et le cas cible élaboré « *TargetCase Req* » et retourne la valeur « true » si tous les inputs du cas source ont leurs correspondants de type « Exact » ou « Subsumes » dans la liste des inputs de la requête (cf. figure 7.8).

```
/*
***fonction qui vérifie si tous les inputs du cas source***
***ont leurs correspondants dans la liste des inputs de la requete ***
*/
boolean VerifyCaseInpTotalMatch (SourceCase Cas, TargetCase Req)
{
    LinpC= Cas.getListeInput() ;
    nbInp = LinpC.getnb();
    LinpR= Req.getListeInput() ;

    int i=0;
    boolean response =true;
    while (i< nbInp)
    {
        if haveCaseInputMatch (LinpC[i], LinpR)
        {
            i ++;
        }
        else
        {
            response =false;
            break;
        }
    }
    return (response);
}
```

Figure 7.8: Algorithme de la fonction VerifyCaseInpTotalMatch

Cette fonction fait appel à une autre fonction appelée *haveCaseInputMatch* qui vérifie si un input d'un cas source possède son correspondant de type « Exact » ou « Subsumes » dans la liste des inputs de la requête. Pour ce faire, elle prend comme paramètres d'entrée un *input inp* du cas et la liste des inputs de la requête *LinpR*. Elle retourne la valeur « true » si elle trouve un correspondant de l'*input inp* de type « Exact » ou « Subsumes » dans la liste *LinpR*. Sinon, elle retourne la valeur « false » (cf. figure 7.9).

```
1  /*
2  ***fonction qui vérifie si un input du cas source***
3  ***a son correspondant dans la liste des inputs de la requete***
4  */
5  boolean haveCaseInputTotalMatch(input inp, list LinpR)
6  {
7      nbInp = LinpR.getnb();
8      int j=0;
9      boolean response =false;
10
11     while (j< nbInp)
12     {
13         if ((TabSimFonc (inp, LinpR[j]) = = "Exact") ||
14         (TabSimFonc (inp, LinpR[j]) = = "Subsumes")
15             {
16                 response=true;
17                 break;
18             }
19         else
20         {
21             j++;
22         }
23     }
24     return (response);
25 }
```

Figure 7.9 : Algorithme de la fonction haveCaseInputMatch

Après avoir exposé les fonctions de vérification des outputs et des inputs, nous présentons la fonction *Create_E_Set* (cf. figure 7.10) qui permet de créer l'ensemble des cas respectant les conditions vérifiées par ces fonctions et qui constituent les cas sources candidats. Cette fonction prend comme paramètres d'entrée le cas cible élaboré « *TargetCase Req* » et la communauté qui lui est associée « *Community Com* ». Elle n'est exécutée que si la fonction *VerifyOutpTotalMatch* est vérifiée. L'ensemble E qu'elle crée regroupe les cas sources de la communauté sélectionnée pour lesquels la valeur des fonctions *VerifyCaseOutpTotalMatch* et *VerifyCaseInpTotalMatch* est « true ».

Quand la fonction *VerifyOutpTotalMatch* n'est pas vérifiée, une notification est renvoyée à l'administrateur et la procédure de découverte se termine après avoir affiché un message d'échec au client.

```
void Create_E_Set (TargetCase Req, Community Com, CaseSet E)
{
    E = NULL;
    if  (VerifyOutpTotalMatch (TargetCase Req, Community Com))
        {
            foreach Case of Com do
            {
                if (VerifyCaseOutpTotalMatch (cas, Req))
                && (VerifyCaseInpTotalMatch (cas, Req))
                    {
                        AddCase(cas, E) ; //E= E U {cas}
                    }
            }
        }
    else
        {
            System.println ("aucun cas correspondant n'existe") ;
            Notify_Admin(); //Transférer la requête a l'administrateur du système
            Exit() ;
        }
}
```

Figure 7.10: Algorithme de la fonction Create_E_Set

7.3.4. Algorithme de calcul de similarité fonctionnelle

Pour le calcul de la similarité fonctionnelle entre un cas source et un cas cible, nous considérons la fonction *FunctionalSimilarity*. Cette fonction compare les parties fonctionnelles d'un cas source et du cas cible, donnés en entrée, tout en parcourant leurs descripteurs et leurs attributs afin de calculer leurs similarités locales. Ces similarités locales seront agrégées par le moyen de la formule de Manhattan afin de générer le degré de similitude fonctionnelle entre le cas source et le cas cible.

Nous appliquons en fait la fonction *FunctionalSimilarity* à chaque cas de l'ensemble E de cas sources candidats afin de calculer sa similarité fonctionnelle vis-à-vis du cas cible. Nous sauvegardons les résultats dans une table nommée **TabSim** qui contient les résultats de calcul de similarité avec les différents cas sources candidats. Cette table comprend trois champs : SimF (similarité fonctionnelle), SimNF (similarité non-fonctionnelle) et SimG (similarité globale). Au départ, les champs de cette structure (**TabSim)** sont initialisés à NULL. Aussi, le traitement correspondant aux mesures fonctionnelles est-il assuré selon l'algorithme de la figure 7.11.

```
/*
***Mesure Fonctionnelle***
*/

float SimFonc=0 ; //Initialiser la SimF à 0

foreach cas of E
{
    // Calculer similarité fonctionnelle SimFonctionnelle
    SimFonc= FunctionalSimilarity (cas, Req);
    // Remplir la valeur SimF dans le champs correspondant de la table TabSim
    TabSim [cas][SimF]=SimFonc;
}
```

Figure 7.11: Algorithme de calcul des mesures fonctionnelles des cas sources

Nous présentons ci-après la fonction *FunctionalSimilarity* en spécifiant le détail des traitements qu'elle effectue à chaque niveau à savoir, les inputs, les outputs, les preconditions et les postconditions comme illustré dans les figures 7.12, 7.13, 7.14, 7.15, 7.16 et 7.17.

```
/*
***fonction de similarité fonctionnelle entre un cas source de l'ensemble E et le cas cible***
*/
float FunctionalSimilarity (SourceCase cas, TargetCase Req, float SubSim, float PlugSim)
{
    float SimF = 2 ; // ds1 et ds6 supposés avoir une correspondance "Exact"
    int NbElements =2 ; // ds1 et ds6 supposés comparés, le nb d'elemens comparés pour le moment est 2

    //Similarité des Inputs
    //Soit getListeInput() la fonction qui nous rend la liste des concepts inputs d'un cas
    LinpC= cas.getlisteInput() ;
    LinpR= Req.get ListeInput() ;
    float SimInp=0 ; // SimInp: la similarité entre les inputs des 2 cas comparés
    int NbInp=0 ; // NbInp: le nb des éléments Inputs comparés

    foreach Input i of LinpC
    {
        mysql-> SELECT i' FROM LinpR HAVING (TabSimFonc(i,i')= "Exact" || "Subsumes");
        if TabSimFonc (i, i') = "Exact"
        {
            SimInp = SimInp + 1 ; //Exact Match
        }
        else
        {
            SimInp = SimInp + SubSim ;
            //SubSim: valeur de mesure de similarité précisée par l'administrateur (expert) pour un subsumes Match
        }
        NbInp++;
        DeleteInput(i', LinpR) ;// Supprimer l'Input i' de la liste LinpR
    }
    SimF=SimF + SimInp;
    NbElements=NbElements + NbInp;
    /*
    **********************fin du traitement pour les Inputs **********************
    */
```

Figure 7.12: Extrait de la fonction FunctionalSimilarity traitant la similarité entre les inputs

```
36
37      //Similarité des Outputs
38      //Soit getListeOutput() la fonction qui nous rend la liste des concepts Outputs d'un cas
39      LOutpC= cas.getlisteOutput) ;
40      LOutpR= Req.get ListeOutput() ;
41      float SimOutput=0 ;
42      int NbOutp=0 ;
43
44          foreach Output o of LOutpR
45          {
46          mysql->SELECT o' FROM LOutpC HAVING (TabSimFonc(o, o')= "Exact" || "Plugin");
47          if TabSimFonc (o,o') = "Exact"
48          {
49              SimOutput = SimOutput + 1 ; //Exact Match
50          }
51          else
52          {
53              SimOutput = SimOutput + PlugSim ;
54              //PlugSim: valeur de mesure de similarité précisée par l'administrateur (expert) pour un Plugin Match
55          }
56          NbOutp ++;
57          DeleteOutput(o', LOutpC) ;// Supprimer l'Output o' de la liste LOutpC;
58          }
59      }
60      SimF=SimF + SimOutnp;
61      NbElements=NbElements + NbOutp;
62      /*
63      ************************fin du traitement pour les Outputs ************************
64      */
```

Figure 7.13 : Extrait de la fonction FunctionalSimilarity traitant la similarité entre les outputs

```
66
67      //Similarité des Preconditions
68      //Soit getListePrec() la fonction qui nous rend la liste des preconditons d'un cas
69      LPrC= cas.getlistPrec() ;
70      LPrR= Req.get ListePrec() ;
71      int nbPrR = LPrR.getnb();
72      int nbPrC = LPrC.getnb();
73      int NbPr= nbPrR + nbPrC;
74
75      float SimPr=0 ;
76
77      if (LPrR!=NULL)
78      {
79          foreach Precondition Pr of LPrR
80          {
81              mysql-> ResPrec= SELECT Pr' FROM LPrC HAVING (TabSimFonc (Pr.Concept, Pr'.Concept) ="Exact");
82              if ResPrec (!=NULL)
83              {
84                  // SimCond fonction qui calcule la similarité entre 2 conditions
85                  SimPr = SimPr + SimCond (Pr, Pr') ;
86              }
87          }
88      }
89      SimF= SimF + SimPr ;
90      NbElements = NbElements + NbPr;
91      /*
92      ************************fin du traitement pour les Preconditions ************************
93      */
```

Figure 7.14: Extrait de la fonction FunctionalSimilarity traitant la similarité entre les préconditions

```
94
95      //Similarité des Postconditions
96      //Soit getListePostc() la fonction qui nous rend la liste des postconditons d'un cas
97      LPostC= cas.getlistPostc() ;
98      LPostR= Req.get ListePostc() ;
99      int nbPostR = LPostR.getnb();
100     int nbPostC = LPPostC.gretnb();
101     int NbPost= nbPostR + nbPostC;
102     float SimPost=0 ;
103
104     if (LPostR!=NULL)
105     {
106       foreach Postcondition Po of LPostR
107         {
108             mysql-> ResPost=SELECT Po' FROM LPostC HAVING (TabSimFonc (Po.Concept, Po'.Concept) ="Exact");
109             if ResPost (!=NULL)
110             {
111                // SimCond fonction qui calcule la similarité entre 2 conditions
112                SimPost = SimPost + SimCond (Po, Po') ;
113             }
114         }
115     }
116     SimF= SimF + SimPost ;
117     NbElements = NbElements + NbPost;
118     /*
119     ************************fin du traitement pour les Postconditions ************************
120     */
```

Figure 7.15: Extrait de la fonction FunctionalSimilarity traitant la similarité entre les postconditions

```
121
122     SimF = SimF/NbElements;
123     return (SimF) ;
124 } // fin fonction FunctionnalSimilarity
```

Figure 7.16: Extrait de la fonction FunctionalSimilarity illustrant le calcul de la similarité finale

La figure 7.17 représente l'algorithme de la fonction SimCond( ) utilisée pour le calcul de la similarité des préconditions et postconditions. Cette fonction prend comme paramètres d'entrée deux conditions *Cond1* et *Cond2* dont les concepts sont équivalents et retourne le degré de similarité entre elles. Nous rappelons qu'une condition est définie par le 5-uplet suivant <C, V, O, U, W> où :

- C : indique le concept en question. C'est un concept de l'ontologie du domaine d'application (par exemple, couleur de la voiture.).
- V : représente l'instance ou la liste d'instances (valeurs) affectées au concept.
- O: indique un opérateur relationnel tels que les opérateurs =, !=, <, >, ≤ ou ≥.

- U : représente l'unité si le concept est mesurable (grandeur quantitative).
- W : représente le poids (weight) de la condition.

```
//fonction de  similarité entre deux conditions
float SimCond (Cond1, Cond2)
{
    float SimConcept = 1; // concepts supposés équivalents
    float SimValeur = 0 ;
    float SimCFinale= 0 ;

    // calculer la similarité des opérateurs et des valeurs puis la similarité finale
    if ((TabSimNFonc (Cond1.Op, Cond2.Op)= = "Exact") && TabSimNFonc (Cond1.Unit, Cond2.Unit) = = Exact))
    {
        //recupérer la liste de valeurs
        List1=Cond1.Value.getlist();
        List2=Cond2.Value.getlist();
        for (i=0; i<List1.width && SimValeur !=1; i++)
            {
                for (j=0; j<List2.width ; j++)
                {
                    if (TabSimNFonc (List1[i], List2[j]) = = "Exact");
                    {
                        SimValeur =1;
                        break;
                    }
                }
            }
        SimCFinale=SimConcept*SimValeur ;
    }
    return (SimCFinale);
}
```

Figure 7.17: Algorithme de la fonction SimCond

7.3.5. Algorithmes de calcul de similarité non-fonctionnelle et globale

Pour le calcul de la similarité non-fonctionnelle, nous appliquons la fonction *NonFunctionalSimilarity* aux cas sources candidats (cas de l'ensemble E) ayant un degré de similarité fonctionnelle **SimF** supérieur ou égal au seuil prédéfini. Nous sauvegardons les résultats obtenus dans le champ **SimNF** de notre table **TabSim**. Le traitement correspondant aux mesures non-fonctionnelles est effectué selon l'algorithme de la figure 7.18.

```
/*
***Traitement Non-Fonctionnel***
*/

float SimNF=0 ; //Initialiser la SimNF à 0
mysql-> E'= SELECT (*) FROM TabSim WHERE SimF>= SeuilF
foreach cas of E'
{
    // Calculer similarité fonctionnelle SimFonctionnelle
    SimNF= NonFunctionalSimilarity (cas, Req);
    // Remplir la valeur SimF dans le champs correspondant de la table TabSim
    TabSim [cas][SimNF]=SimNF;
}
```

Figure 7.18: Algorithme de calcul des mesures non-fonctionnelles des cas sources

Nous présentons ci-après la fonction *NonFunctionalSimilarity* utilisée au niveau de l'algorithme de calcul des similarités non-fonctionnelles. Cette fonction prend comme paramètres d'entrée un cas source parmi ceux sélectionnés en plus du cas cible (requête) et retourne le degré de similarité non-fonctionnelle, et ce, en appariant les propriétés non-fonctionnelles des deux cas (cf. figure 7.19).

```
/* ***fonction de similarité non fonctionnelle entre un cas source et le cas cible (Req)*** */
float NonFunctionalSimilarity (cas,Req)
{
    float Wsom = 0; // Somme des poids des Assertions
    float SimNF=0; // Non-Functional Similarity
    float SimAssert=0; // SimAssert: similarity between 2 Assertions
    float Sim=0; // Sim: similarity between 2 Assertion Lists
    int NbElements=0; // Number of compared elements
    foreach PolicySubject PS from {Service, EndPoint, Operation, InMsg, OutMsg, FaultMsg, Binding}
    {
        if (PS.AssertionList(Req)!=NULL)
            {
                foreach Assertion A of PS.AssertionList(Req)
                {
                    mysql-> Res=Select A' FROM PS.AssertionList(Cas)
                         ->  HAVING(TabSimNFonc(A.concept,A'.concept)== "Exact"));
                    if (Res!=NULL)
                    {
                       SimAssert = SimiCond (A, A');
                       Sim = Sim + (SimAssert * (A.weight));
                       Wsom = Wsom + A.weight;
                    }
                }
            }
        NbElements+= PS.AssertionListNumb(Req);
    }
    SimNF = Sim / NbElements;
    if (Wsom >1)
    {
        SimNF = SimNF / Wsom;
    }
    Return (SimNF);
}
```

Figure 7.19: Fonction NonFunctionalSimilarity calculant la similarité non-fonctionnelle

Finalement, pour le calcul de la similarité globale, nous considérons l'algorithme donné dans la figure 7.20. Cet algorithme sélectionne d'abord les cas sources satisfaisant les besoins

fonctionnels et non-fonctionnels du client parmi les cas de l'ensemble E (cas sources candidats) et ensuite calcule leur mesure de similarité globale vis-à-vis du cas cible.

```
/*
***Traitement Final***
*/

float SimG=0 ; //Initialiser la SimG à 0
mysql-> FinalSet =  SELECT (*) FROM TabSim
    ->              WHERE SimF>= SeuilF
                    AND  SimNF>=SeuilNF

foreach cas of FinalSet
{
// Calculer similarité globale conformément à la formule ci dessus
SimG= (SimF+SimNF)/2 ;
// Remplir la valeur SimG dans le champs correspondant de la table TabSim
TabSim [cas][SimG]=SimG;
}// fin traitement Final
```

Figure 7.20: Algorithme de calcul de similarité globale

7.4. IHM du système CBR4WSD

Dans cette section, nous donnons un aperçu de la hiérarchie des interfaces homme-machine que nous avons réalisées et qui correspondent à des fonctionnalités spécifiques de notre système.

L'espace administrateur regroupe les interfaces relatives au chargement d'une ontologie et la classification de ses concepts, à l'alimentation de la base de cas par l'ajout d'un cas source en ses parties problème et solution, à la configuration des règles et des seuils de similarité et à la maintenance de la base de cas. Le scénario d'utilisation de ces interfaces commence par l'authentification de l'administrateur, qui une fois authentifié, accède à l'écran d'accueil à partir duquel il choisit de lancer la fonctionnalité d'administration ou de configuration souhaitée.

L'espace client regroupe les interfaces relatives à la formalisation et le lancement d'une requête de découverte de services Web et à l'exploitation des résultats retournés.

Nous présentons par la suite les IHMs de l'application réalisée regroupées par fonctionnalités.

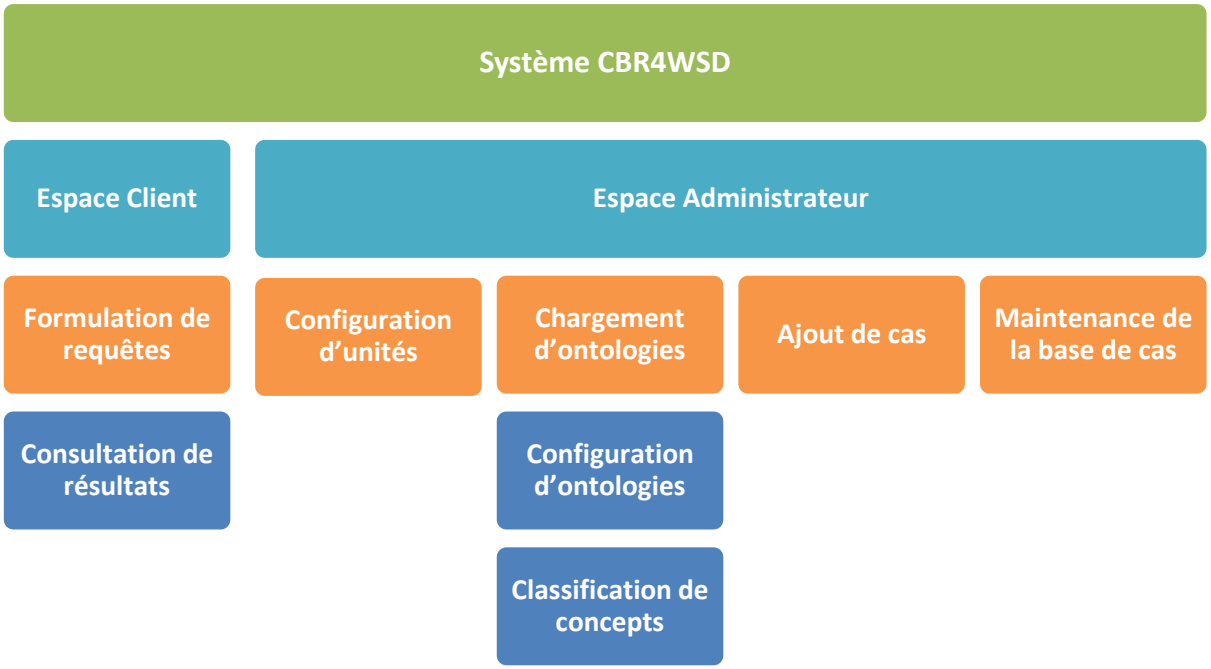


Figure 7.21: Hiérarchie des IHM dans le système CBR4WSD

7.5. Synthèse

La remémoration représente la phase cruciale de notre système CBR4WSD. Cette phase a été réalisée en deux processus : online et offline. Les tableaux 7.2 et 7.3 résument les algorithmes et les fonctions implémentées dans chacun de ces processus.

Processus de remémoration Offline		
Algorithme	**Fonctions intégrées**	**Rôle de la fonction**
Algorithme de calcul de similarité offline	*créerMatrice*	Créer la matrice de similarité entre concepts ontologiques.

Tableau 7.2: Tableau représentant l'algorithme et la fonction correspondante implémentée dans le processus offline

Le tableau 7.3 synthétise les algorithmes et les fonctions implémentées dans le processus online.

Processus de remémoration Online		
Algorithme	**Fonctions intégrées**	**Rôle de la fonction**
Algorithme de vérification de découvrabilité	*Verify-Community*	Vérifier si le descripteur de la communauté dans le cas cible est non nul
	VerifyOutpTotalMatch et HaveOutputMatch	Vérifier si tous les outputs de la requête ont leurs correspondants de type Exact ou Plugin dans la liste des outputs de la communauté.
Algorithme de sélection des cas sources candidats	*VerifyCaseOutpTotalMatch et HaveCaseOutpTotalMatch*	Vérifier si tous les outputs de la requête ont leurs correspondants de type Exact ou Plugin dans la liste des outputs du cas source.
	VerifyCaseInpTotalMatch et HaveCaseInpTotalMatch	Vérifier si tous les inputs du cas source ont leurs correspondants de type Exact ou Subsumes dans la liste des inputs de la requête.
	Create_E_Set	Créer l'ensemble de cas sources candidats
Algorithme de calcul de similarité fonctionnelle	*FunctionalSimilarity et SimCond*	Calculer la similarité fonctionnelle entre un cas source et cas cible
Algorithme de calcul de similarité non-fonctionnelle	*NonFunctionalSimilarity et SimCond*	Calculer la similarité non-fonctionnelle entre un cas source et cas cible
Algorithme de calcul de similarité globale		Calculer la similarité globale entre un cas source et cas cible

Tableau 7.3: Tableau récapitulatif des algorithmes et des fonctions implémentées dans le processus online

Troisième partie

Expérimentation

Chapitre 8

Expérimentation et validation

SOMMAIRE

8.1. Introduction

Ce chapitre est consacré à la validation de notre approche. Pour celà nous procédons à une expérimentation de notre système, une évaluation de ses performances et une évaluation de ses principes directeurs. Notre cas d'expérimentation relève du domaine du e-tourisme, en particulier, les services de réservation de billets d'avion. Il s'appuie sur une base de 60 cas.

Nous commencons par présenter le scénario de notre expérimentation qui met en evidence l'interface utilisateur. Nous abordons ensuite le bilan de cette expérimentation en comparant les résultats attendus avec les résultats obtenus. Nous terminons par une évaluation sommaire de la complexité de nos algorithmes pour montrer l'impact de nos deux principes directeurs : communauté de services et priorité à l'aspect fonctionnel du service Web.

8.2. Scénario d'expérimentation

Pour l'expérimentation de notre système, nous avons considéré une ontologie de domaine que nous dénommons « e-tourism.owl » en vue d'annoter les propriétés fonctionnelles d'un cas de service. Nous avons également considéré une ontologie dénommée « PNF.owl » pour annoter des propriétés non-fonctionnelles comme celles relatives à la qualité de service. Nous avons élaboré 60 cas pour alimenter la base de cas de services Web. Pour évaluer nos algorithmes, nous avons opté pour la construction d'une requête représentative permettant de tester les différentes situations : cette requête relève d'une communauté regroupant 27 cas parmi lesquels une quinzaine satisfait ses exigences fonctionnelles et 9 d'entre eux répondent à ses exigences non-fonctionnelles. 7 cas seront proposés au client comme solution de référence.

Les ontologies de référence :

Nous illustrons graphiquement ces deux ontologies (e-tourisme et PNF) que nous avons implémentées au moyen du logiciel « Protege », dans les figures 8.1 et 8.2. Les fichiers de description OWL de ces deux ontologies seront donnés en annexe.

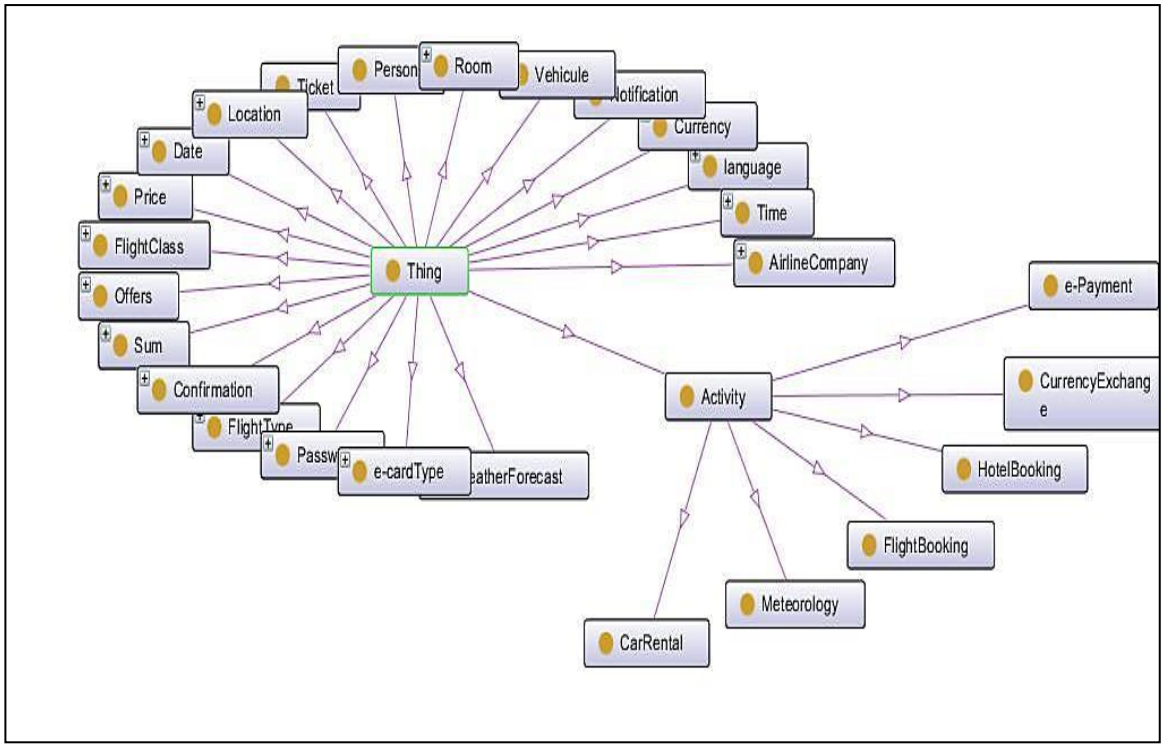


Figure 8.1: Extrait de l'ontologie « e-tourism.owl »

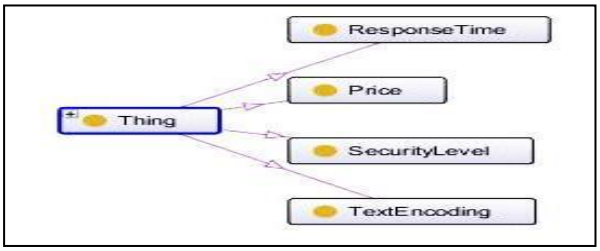


Figure 8.2: Extrait de l'ontologie « PNF.owl »

Nous notons que l'utilisation du système CBR4WSD n'est pas restreinte à une ontologie de domaine spécifique. Ce dernier, tel qu'il a été conçu, reste ouvert à l'utilisation de différentes ontologies. Il donne la possibilité au client de choisir, parmi une liste extensible de domaines prévus, le domaine métier qu'il cible et charge automatiquement l'ontologie correspondante.

Mode offline : Première étape configuration des ontologies

Nous avons alors démarré la phase d'expérimentation par le chargement des ontologies OWL e-tourism (cf. figure 8.3) et PNF, puis la spécification des seuils de similarité fonctionnelle, non-fonctionnelle et globale ainsi que celle des valeurs accordées au degré de similarité Subsumes et Plugin comme illustré par la figure 8.4.

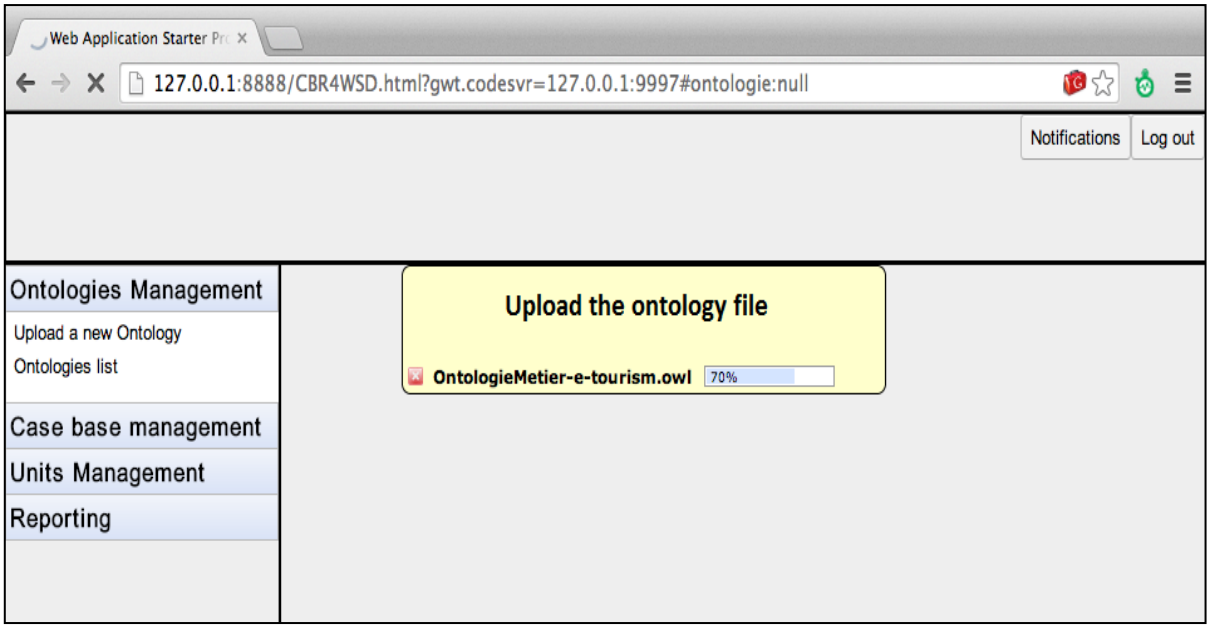


Figure 8.3: Chargement d'une ontologie de domaine.

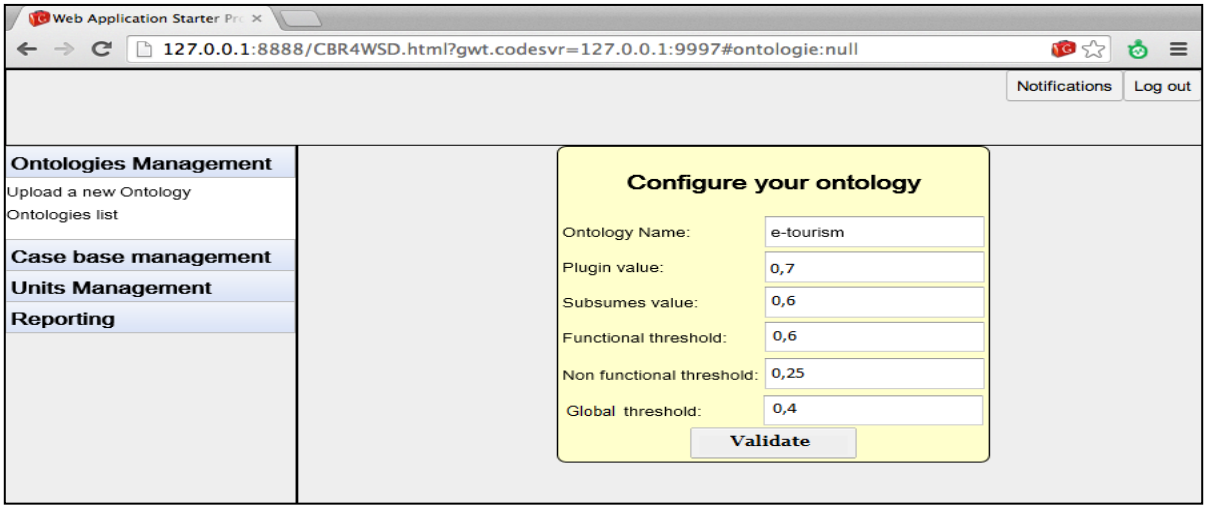


Figure 8.4: Configuration des seuils et valeurs de similarité par ontologie de domaine.

Mode offline : Deuxième étape alimentation de la base de cas

Après avoir lancé le calcul de similarité entre les différents concepts des ontologies chargées et ce toujours en mode offline, nous avons procédé à l'alimentation de la base de cas par un échantillon de soixante cas sources en introduisant leurs parties problème et solution comme illustré par l'exemple des figures 8.5 et 8.6.

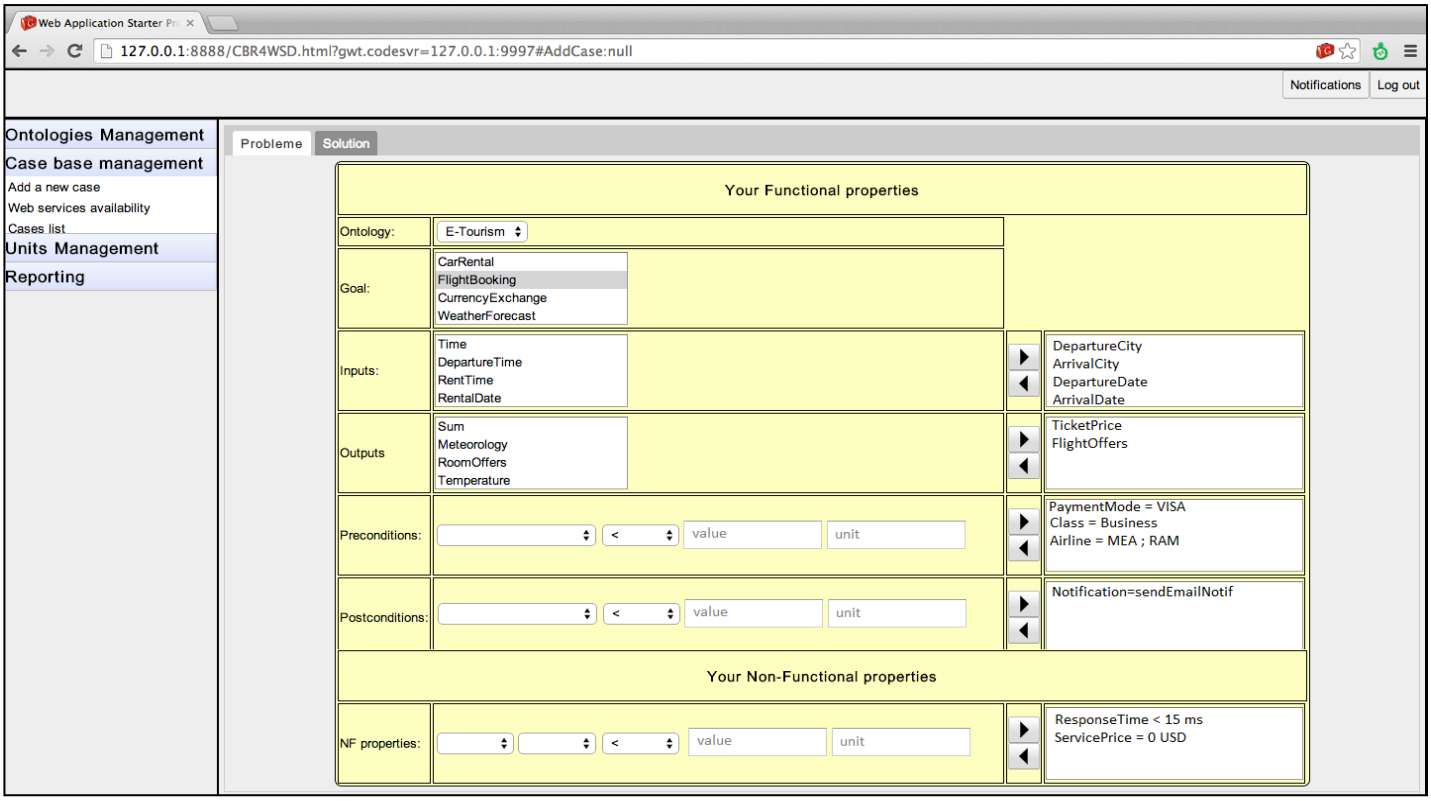


Figure 8.5 : Introduction de la partie problème d'un cas source

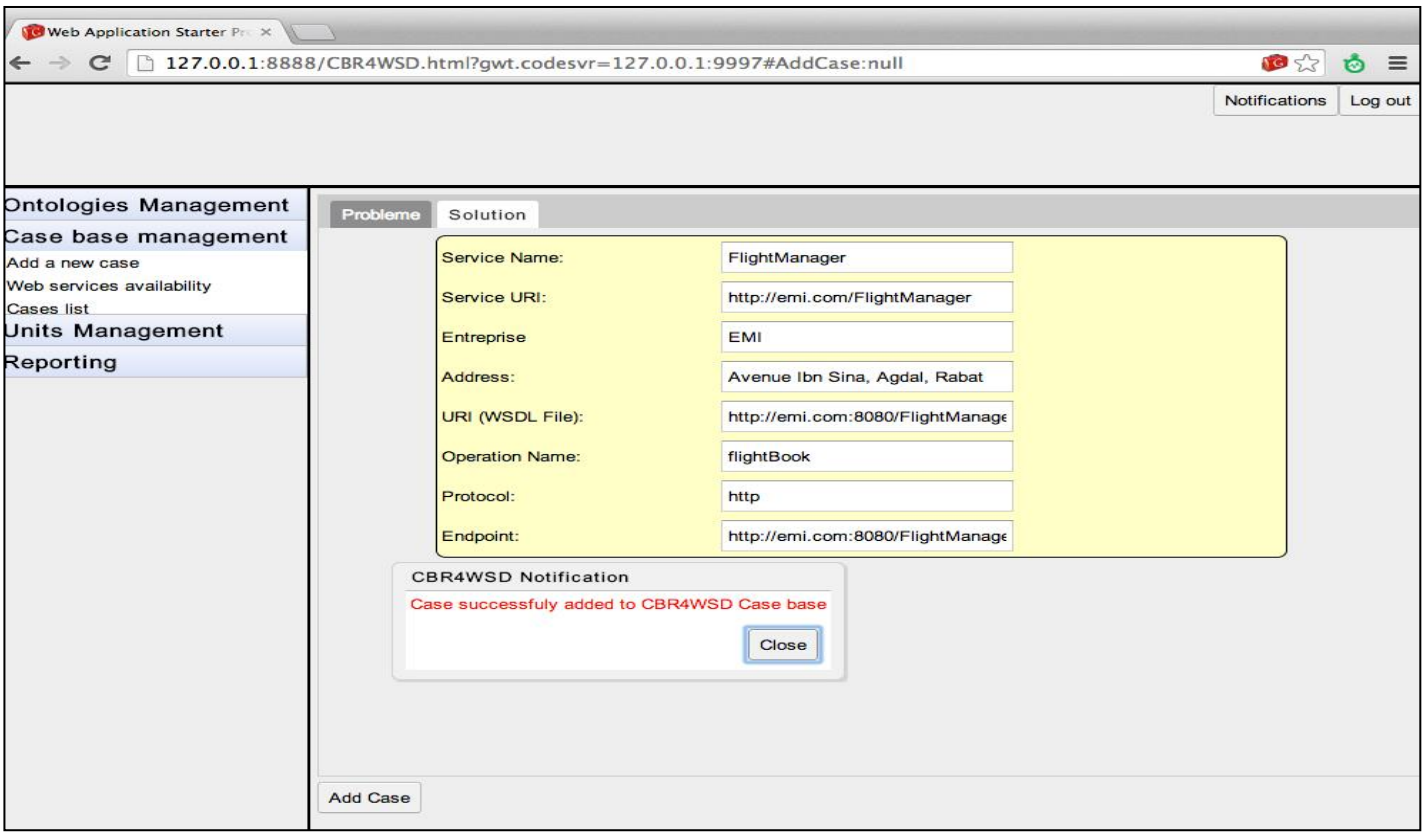


Figure 8.6: Introduction de la partie solution d'un cas source

Par ailleurs, il est à noter que les cas considérés se répartissent en six communautés de services ciblant différents « goals » de l'ontologie e-tourism à savoir, «FlightBooking », « HotelBooking », « CarRental », « e-Payment », « CurrencyExchange » et « Meteorology ». Le tableau 8.1 illustre la répartition de ces cas dans les six communautés.

Identifiant de la communauté	Goal	Nombre de cas sources
Com1	FlightBooking	27
Com2	HotelBooking	10
Com3	CarRental	10
Com4	e-Payment	2
Com5	Meteorology	6
Com6	CurrencyExchange	5

Tableau 8.1 : Répartition des cas sources sur les six communautés de l'échantillon de test

Mode online : Première étape formulation de la requête

Finalement, pour le test de la performance des résultats produits par notre système, nous avons considéré une requête de découverte de services de réservation de billet d'avions (cf. figure 8.7). Cette requête vise à chercher un service qui prend en entrée quatre paramètres : ville de départ (*DepartureCity)*, ville d'arrivée (*ArrivalCity)*, date de départ (*DepartureDate)* et date d'arrivée (*ArrivalDate)* et qui retourne deux paramètres : le prix du billet (*TicketPrice)* et les offres possibles (*FlightOffers).*

La requête exige que le service découvert admette un paiement électronique par carte « VISA » ou « American Express ». Cette contrainte est exprimée par la précondition *PaymentMode={VISA, American Express}*. Le service cherché doit aussi permettre la notification de l'utilisateur par email (pour confirmer ou infirmer la réservation). Cette contrainte est exprimée par la postcondition *Notification=sendEmailNotification*.

En plus, la requête en question cherche également un service de préférence non payant (*ServicePrice=0 USD*) et qui répond dans un temps inférieur à 15 secondes (*ResponseTime <15 ms*). Un poids de 0,5 est affecté à chacune de ces préférences. La liste des résultats obtenus est illustrée dans la figure 8.8.

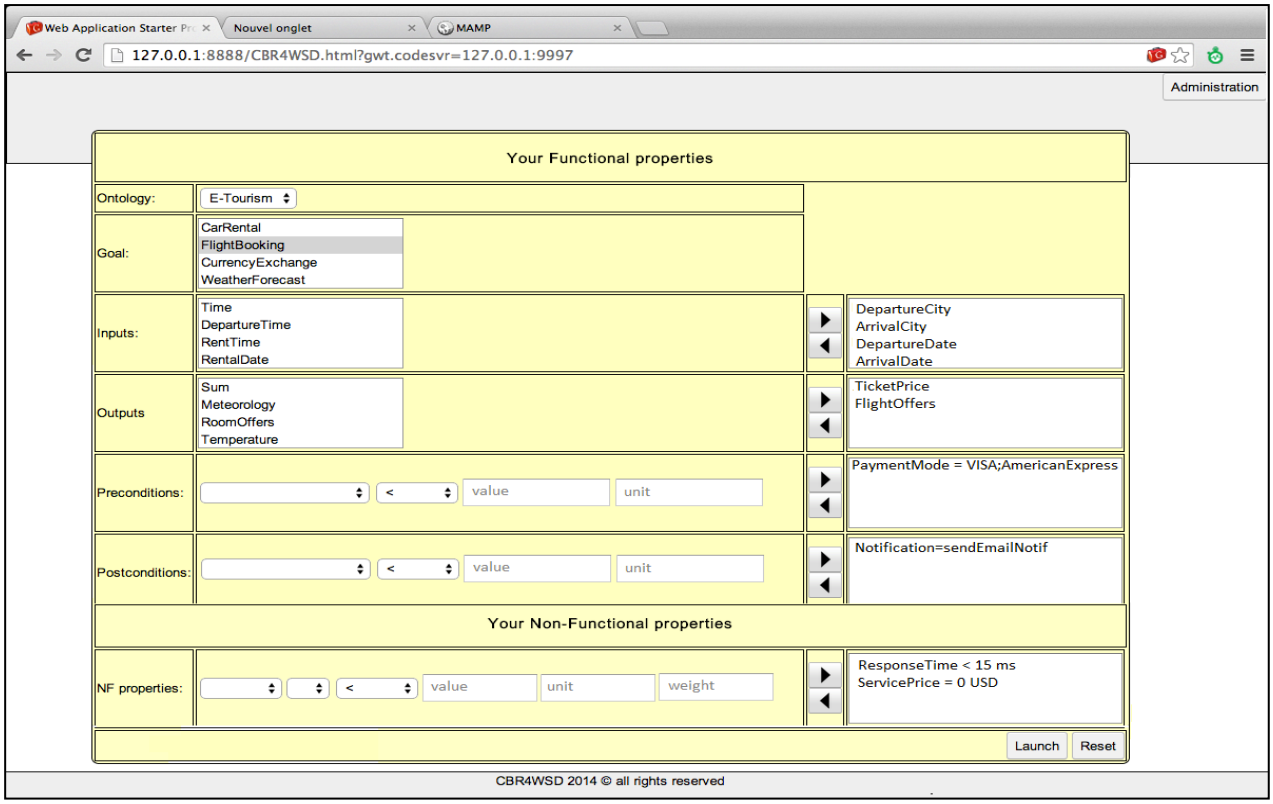


Figure 8.7: Requête de découverte d'un service de réservation de billets d'avion.

Mode online : Deuxième étape consultation des résultats à la requête

Les résultats affichés dans la figure 8.8 font référence à des services qui répondent aux besoins fonctionnels et non-fonctionnels exprimés dans la requête cible. Un clic sur l'un des résultats permet d'accéder aux détails décrivant la solution à laquelle il fait référence ainsi qu'à ses degrés de satisfaction fonctionnelle, non-fonctionnelle et globale.

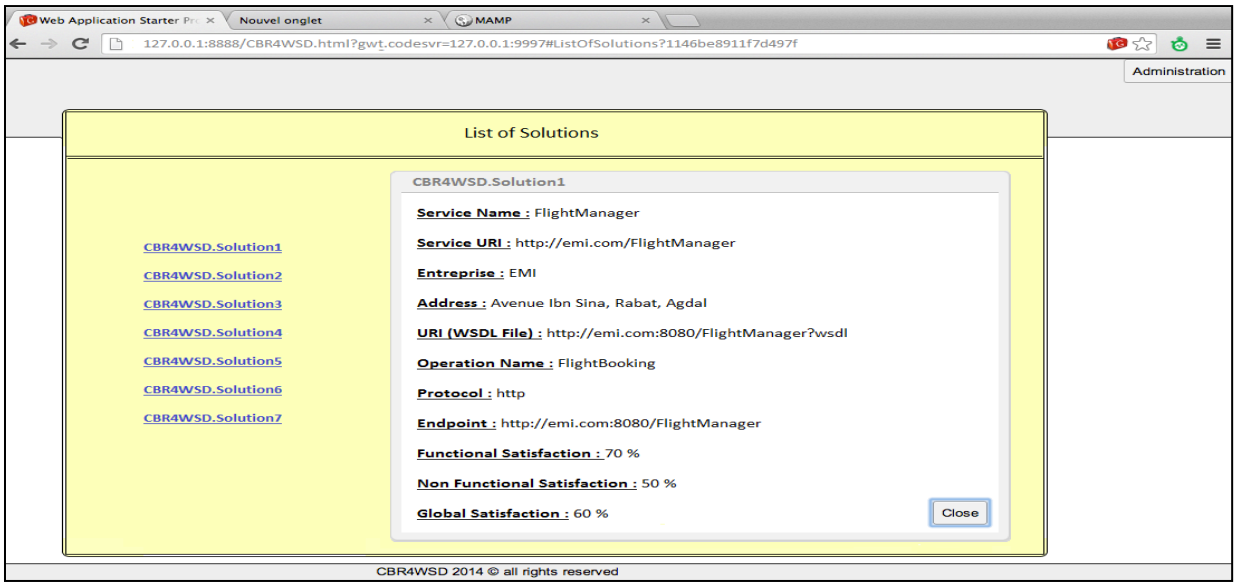


Figure 8.8: Liste des résultats obtenus et détails d'un exemple de solution

8.3. Bilan d'expérimentation

Pour évaluer et valider notre approche, nous l'avons appliqué à un échantillon de soixante cas sources comme noté auparavant. Le processus de découverte a été lancé pour la requête représentée dans la figure 8.7. En effet, au début du traitement, le composant « élaborateur du cas cible » doit associer à cette requête la communauté de services « Com1 » qui correspond au goal « FlightBooking ». De ce fait, 27 cas devraient être retenus pour l'étape suivante où seront appliquées les règles de vérification de la découvrabilité de la requête et de sélection des cas sources candidats. Comme la liste des outputs de la communauté concernée « Com1» regroupe les concepts TicketPrice, Price, Offers, FlightOffers et LowestFairOffer, la requête devrait être considérée comme découvrable (voir signature de « Com1 » en annexe).

Le système poursuit par la suite son traitement par la création de l'ensemble E regroupant les cas sources de la communauté qui répondent fonctionnellement au cas cible. En effet, le cas source candidat doit fournir au minimum les outputs du cas cible. En plus, il ne doit pas exiger plus d'inputs que ceux spécifiés par le client. Aussi, relativement aux cas considérés dans l'échantillon de test, douze cas sources figurant parmi les vingt-sept cas de la communauté « Com1 », ne devraient pas être considérés comme éléments de l'ensemble E des cas candidats. Les exemples des tableaux 8.2 et 8.3 illustrent des cas sources devant être écartés.

Exemple de cas écarté (input de plus)			
Goal	bookFlight	---	---
Input	$ds^{C}_{2.1}$: DepartureDate	Operation Policy	$ds^{C}_{8,1}$: ResponseTime
	$ds^{C}_{2.2}$: ArrivalCity		$ds^{O}_{8,1}$:<
	$ds^{C}_{2.3}$: DepartureCity		$ds^{V}_{8,1}$: 15
	$ds^{C}_{2.4}$: ArrivalDate		$ds^{U}_{8,1}$: ms
	$ds^{C}_{2.5}$: Airport		
Output	$ds^{C}_{3.1}$: Price		$ds^{C}_{8,2}$: ServicePrice
	$ds^{C}_{3.2}$: Offers		$ds^{O}_{8,2}$:=
Précondition	ds^{C}_{4}: PaymentMode		$ds^{V}_{8,2}$: 0
	ds^{V}_{4}:{VISA; American Express}		$ds^{U}_{8,2}$: USD
	ds^{O}_{4} :=	---	---
Postcondition	ds^{C}_{5}: Notification	---	---
	ds^{V}_{5}: sendEmailNotif	---	---
	ds^{O}_{5} :=	---	---
ServCom	Com1	---	---

Tableau 8.2 : Exemple de cas source exigeant un input de plus par rapport à la requête

Exemple de cas écarté (output de moins)			
Goal	bookFlight	---	---
Input	$ds^{C}_{2.1}$: DepartureDate	Operation Policy	$ds^{C}_{8,1}$: ResponseTime
	$ds^{C}_{2.2}$: ArrivalCity		$ds^{O}_{8,1}$:<
	$ds^{C}_{2.3}$: DepartureCity		$ds^{V}_{8,1}$: 15
	$ds^{C}_{2.4}$: ArrivalDate		$ds^{U}_{8,1}$: ms
Output	$ds^{C}_{3.2}$: Offers		$ds^{C}_{8,2}$: ServicePrice
			$ds^{O}_{8,2}$:=
Précondition	ds^{C}_{4}: PaymentMode		$ds^{V}_{8,2}$: 0
	ds^{V}_{4}:{AmericanExpress}		$ds^{U}_{8,2}$: USD
	ds^{O}_{4} :=	---	---
Postcondition	ds^{C}_{5}: Notification	---	---
	ds^{V}_{5}: sendEmailNotif	---	---
	ds^{O}_{5} :=	---	---
ServCom	Com1	---	---

Tableau 8.3: Exemple de cas source n'offrant pas un output exigé par la requête

8.3.1. Les résultats attendus (calculés manuellement) :

Après avoir identifié l'ensemble des cas sources vérifiant les besoins fonctionnels du client en termes d'outputs et d'inputs, le processus de découverte doit procéder au calcul de la similarité fonctionnelle de chacun de ces cas sources avec le cas cible. Le tableau 8.4 illustre le contenu de la table **TabSim** devant être obtenu à la fin de la phase de calcul des mesures de similarité fonctionnelle. Les cas dont la SimF est notée en vert ont un degré de similarité fonctionnelle supérieure ou égale au seuil de similarité fonctionnelle prédéfinie par l'administrateur (0,6). Ces cas seront considérés à l'étape suivante de calcul des mesures de similarité non-fonctionnelle.

Cas sources	SimF	SimNF	SimG
cas_{11}	0,79	NULL	NULL
cas_{12}	0,7	NULL	NULL
cas_{13}	0,62	NULL	NULL
cas_{14}	0,58	NULL	NULL
cas_{15}	0,56	NULL	NULL
cas_{16}	0,6	NULL	NULL
cas_{17}	0,56	NULL	NULL
cas_{31}	0,56	NULL	NULL
cas_{32}	0,6	NULL	NULL
cas_{33}	0,56	NULL	NULL
cas_{34}	0,58	NULL	NULL
cas_{41}	0,7	NULL	NULL
cas_{42}	0,7	NULL	NULL
cas_{43}	0,56	NULL	NULL
cas_{44}	0,7	NULL	NULL

Tableau 8.4: Etat de la table TabSim en fin de calcul des mesures fonctionnelles

Le tableau 8.5 illustre le contenu de la table **TabSim** devant être obtenu à la fin de la phase de calcul des mesures non-fonctionnelles. Les cas dont la **SimNF** est notée en vert ont un degré de similarité non-fonctionnelle supérieure ou égale au seuil de similarité non-fonctionnelle prédéfinie par l'administrateur (0,25).

Cas sources	SimF	SimNF	SimG
cas_{11}	0,79	0	NULL
cas_{12}	0,7	0,25	NULL
cas_{13}	0,62	0,5	NULL
cas_{14}	0,58	NULL	NULL
cas_{15}	0,56	NULL	NULL
cas_{16}	0,6	0,5	NULL
cas_{17}	0,56	NULL	NULL
cas_{31}	0,56	NULL	NULL
cas_{32}	0,6	0,25	NULL
cas_{33}	0,62	0	NULL
cas_{34}	0,58	NULL	NULL
cas_{41}	0,7	0,25	NULL
cas_{42}	0,7	0,5	NULL
cas_{43}	0,56	NULL	NULL
cas_{44}	0,7	0,5	NULL

Tableau 8.5: Etat de la table TabSim en fin de calcul des mesures non-fonctionnelles

Lors de l'étape finale où la mesure de similarité globale (MSG) est calculée, seuls les cas ayant des degrés de similarité fonctionnelle et non-fonctionnelle supérieurs ou égaux respectivement au seuil de similarité fonctionnelle (0,6) et au seuil de similarité non-fonctionnelle (0,25) seront considérés. Le tableau 8.6 illustre le contenu de la table **TabSim** devant être obtenu à la fin de la phase de calcul des mesures globales. Les cas dont la **SimG** est notée en vert ont un degré de similarité globale supérieure ou égale au seuil de similarité globale prédéfinie par l'administrateur (0,4). Ce sont exactement les solutions correspondant à ces cas qui ont été sélectionnées par notre système (cf. figure 8.8).

Cas sources	SimF	SimNF	SimG
cas_{11}	0,79	0	NULL
cas_{12}	0,7	0,25	0,475
cas_{13}	0,62	0,5	0,56
cas_{14}	0,58	NULL	NULL
cas_{15}	0,56	NULL	NULL
cas_{16}	0,6	0,5	0,55
cas_{17}	0,56	NULL	NULL
cas_{31}	0,56	NULL	NULL
cas_{32}	0,6	0,25	0,425
cas_{33}	0,62	0	NULL
cas_{34}	0,58	NULL	NULL
cas_{41}	0,7	0,25	0,475
cas_{42}	0,7	0,5	0,6
cas_{43}	0,56	NULL	NULL
cas_{44}	0,7	0,5	0,6

Tableau 8.6 : Etat de la table TabSim en fin de calcul des mesures globales

Nous allons comparer ces résultats calculés manuellement avec ceux que nous a fournis le système CBR4WSD.

8.3.2. Evaluation des résultats obtenus par CBR4WSD

Pour évaluer la performance d'un système en termes de qualité des résultats, la différence entre le résultat attendu et le résultat obtenu doit être mesurée. Pour ce faire, un ensemble d'indicateurs de mesure de performances doit être utilisé [Nakache et Métais, 2005]. Nous commençons par identifier nos indicateurs d'évaluation de la performance avant de les appliquer à notre expérimentation.

8.3.2.1 Les indicateurs d'évaluation

Nous considérons six indicateurs d'évaluation de performance : rappel, silence, précision, bruit, pertinence et F-Mesure.

Indicateurs Rappel et Silence : les bons cas sont-ils tous sélectionnés ?

Le rappel est défini par le nombre de cas sources pertinents retrouvés au regard du nombre de cas sources pertinents contenus dans la base de cas. En fait, lorsque le client interroge la base de cas, il souhaite obtenir toutes les solutions qui pourraient répondre à sa requête. Le taux de rappel devrait alors être élevé. Par contre, si le système possède de nombreux cas sources pertinents et la plupart d'entre eux ne sont pas fournis par le système, le taux du silence serait

élevé. Le silence s'oppose alors au rappel (rappel + silence = 1). Les formules de calcul du rappel et du silence sont présentées ci-après :

$$\textbf{Rappel} = \frac{\textit{nombre de cas sources pertinents extraits}}{\textit{nombre total de cas sources pertinents}}$$

$$\textbf{Silence} = \frac{\textit{nombre de cas sources pertinents non extraits}}{\textit{nombre total de cas sources pertinents}}$$

Indicateurs Précision et Bruit : Les cas sélectionnés sont-ils tous bons ?

La précision est le taux du nombre de cas sources pertinents retrouvés par rapport au nombre de cas sources total retrouvés par le système. En fait, lorsque le client interroge la base de cas, il souhaite que les cas sources proposés en réponse à sa requête soient pertinents. Les cas sources non pertinents retournés constituent du bruit. La précision s'oppose alors au bruit (précision + bruit = 1). Si elle est élevée, cela signifie que peu de cas sources non pertinents sont proposés par le système et que ce dernier peut être considéré comme "précis". Les formules de calcul de la précision et du bruit sont présentées ci-après :

$$\textbf{Précision} = \frac{\textit{nombre de cas sources pertinents extraits}}{\textit{nombre total de cas sources extraits}}$$

$$\textbf{Bruit} = \frac{\textit{nombre de cas sources non pertinents extraits}}{\textit{nombre total de cas sources extraits}}$$

Indicateur d'erreur : Proportion des cas erronés

L'erreur est le taux de la somme des cas pertinents non retrouvés et des cas non pertinents retrouvés, par rapport au nombre total de cas sources extraits. Elle est calculée par la formule suivante :

$$\textbf{Erreur} = \frac{\textit{nombre de cas sources pertinents non extraits} + \textit{nombre de cas sources non pertinents extraits}}{\textit{nombre total de cas sources extraits}}$$

Une mesure populaire qui combine la précision et le rappel est leur pondération, nommée F-mesure. Elle est calculée par la formule suivante :

$$\textbf{F-Mesure} = \frac{2 * Précison * Rappel}{Précision * Rappel}$$

8.3.2.2 Evaluation des résultats à la requête test

En examinant les différents cas de l'échantillon de test considéré, nous avons identifié les cas pertinents répondant à la requête soumise. Le tableau 8.7 présente un récapitulatif des résultats obtenus selon leur pertinence. La figure 8.9 expose ensuite les indicateurs de performance calculés sur la base de ces résultats.

	Pertinents	Non pertinents	Total
Cas sources extraits	7	0	7
Cas sources non-extraits	1	7	8
Total	8	7	15

Tableau 8.7 : Récapitulatif des résultats obtenus

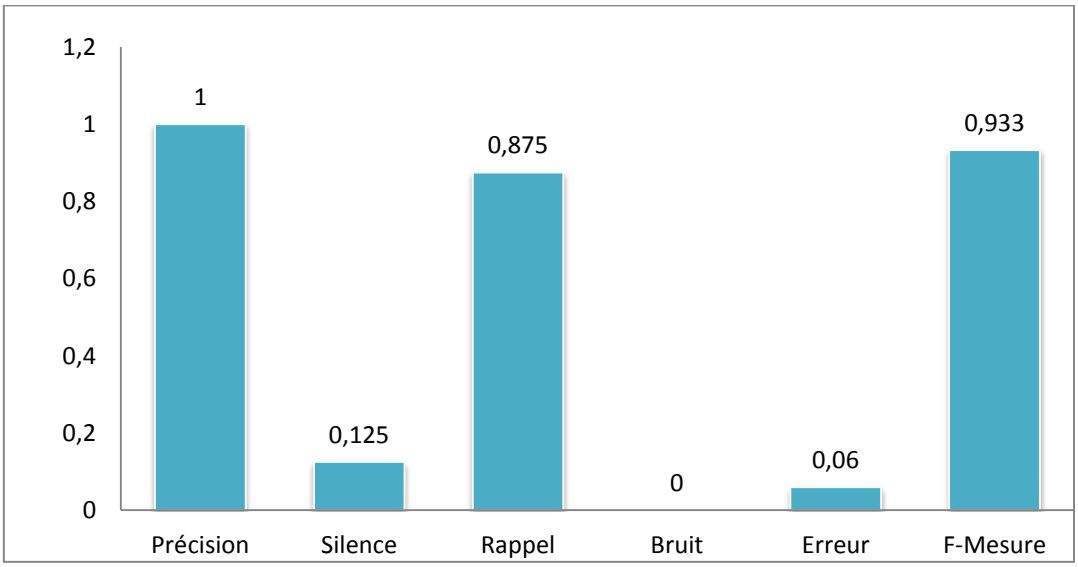


Figure 8.9: Graphe des indicateurs de performance de la qualité des résultats

Un système de découverte de services Web parfait fournira des réponses dont la précision et le rappel sont égaux à 1 (l'algorithme trouve la totalité des services pertinents - rappel - et ne fait aucune erreur - précision). Dans la réalité, les algorithmes de recherche sont plus ou

moins précis, et plus ou moins performants. Il serait possible d'obtenir un système très précis (par exemple, un système ayant un taux de précision de 0,99), mais peu performant (par exemple, un système dont le taux de rappel est de 0.10 qui signifie que le système n'a trouvé que 10 % des réponses possibles). Dans le même contexte, un algorithme dont le rappel est fort (par exemple, un système dont le taux de rappel est de 0,99, soit un système qui retrouve la quasi-totalité des solutions pertinentes), mais dont la précision est faible (par exemple, un système dont le taux de précision est de 0.10) fournira en guise de solutions de nombreuses solutions erronées en plus de celles pertinentes. Le système correspondant sera donc difficilement exploitable.

D'après les résultats de la figure 8.9, nous constatons que notre système peut être qualifié de précis (précision=1) et performant (rappel=0,875). Néanmoins, le degré de silence (0,125) indique que certains cas sources pertinents n'ont pas figuré dans la liste des réponses. C'est le cas du cas source $\boldsymbol{cas_{11}}$ (cf. tableau 8.6) répondant parfaitement aux propriétés fonctionnelles de la requête mais qui a été écarté à cause de sa similarité non-fonctionnelle nulle. Les détails ce cas sont présentés dans le tableau 8.8.

$\boldsymbol{cas_{11}}$			
Goal	bookFlight	---	---
Input	$\boldsymbol{ds_{2.1}^{C}}$: DepartureDate	Operation Policy	$\boldsymbol{ds_{8,1}^{C}}$: ResponseTime
	$\boldsymbol{ds_{2.2}^{C}}$: ArrivalCity		$\boldsymbol{ds_{8,1}^{O}}$ **:** <
	$\boldsymbol{ds_{2.3}^{C}}$: DepartureCity		$\boldsymbol{ds_{8,1}^{V}}$: 10
	$\boldsymbol{ds_{2.4}^{C}}$: ArrivalDate		$\boldsymbol{ds_{8,1}^{U}}$ **:** ms
Output	$\boldsymbol{ds_{3.2}^{C}}$: Offers		---

Précondition	$\boldsymbol{ds_{4}^{C}}$: PaymentMode		---
	$\boldsymbol{ds_{4}^{V}}$:{VISA }		---
	$\boldsymbol{ds_{4}^{O}}$ **:=**	---	---
Postcondition	---	---	---
	---	---	---
	---	---	---
ServCom	Com1	---	---

Tableau 8.8: Exemple de cas pertinent écarté de la liste des résultats retournés au client

Ce cas source représente un service de réservation de billets qualifié d'un temps de réponse inférieur à 10 ms et aurait dû être par conséquent sélectionné car il satisfait le critère du client

qui exige un temps de réponse inférieur à 15 ms. Or, le système ne le considère pas similaire : il lui a accordé une similarité égale à 0 car il se limite à une comparaison stricte des valeurs numériques. Ceci était la cause essentielle de la non-considération de ce cas lors du calcul de similarité globale et par la suite son élimination.

8.3.2.3 Validation de l'utilisation des communautés

Lors de nos expérimentations, nous nous sommes aussi intéressés au temps de réponse. Pour cela, nous avons examiné trois échantillons de cas sources de tailles différentes en utilisant toujours la requête de la figure 8.7.

Nous notons que, les cas considérés dans chaque échantillon se répartissent en six communautés de services ciblant différents « goals » de l'ontologie e-tourism à savoir, «FlightBooking », « HotelBooking », « CarRental », « e-Payment », « CurrencyExchange » et « Meteorology ». Le tableau 8.9 illustre la répartition des cas dans les six communautés considérées pour chaque échantillon.

	Identifiant de la communauté	**Goal**	**Nombre de cas sources**	**Nombre de cas sources candidats**
Echantillon 1 (60 cas)	Com1	FlightBooking	27	15
	Com2	HotelBooking	10	0
	Com3	CarRental	10	0
	Com4	e-Payment	2	0
	Com5	Meteorology	6	0
	Com6	CurrencyExchange	5	0
Echantillon 2 (40 cas)	Com1	FlightBooking	27	15
	Com2	HotelBooking	2	0
	Com3	CarRental	2	0
	Com4	e-Payment	1	0
	Com5	Meteorology	3	0
	Com6	CurrencyExchange	5	0
Echantillon 3 (60 cas)	Com1	FlightBooking	10	10
	Com2	HotelBooking	27	0
	Com3	CarRental	10	0
	Com4	e-Payment	2	0
	Com5	Meteorology	6	0
	Com6	CurrencyExchange	5	0

Tableau 8.9 : Répartition des cas dans trois échantillons de test

Le traitement de la requête considérée cible précisément la communauté « Com1 » pour en découvrir les cas similaires. Ainsi, pour la vérification des règles de découvrabilité, le système examine 27 cas sources dans l'échantillon 1, 27 cas sources dans l'échantillon 2 et 10 cas sources dans l'échantillon 3. Seuls les cas candidats seront considérés lors du calcul de similarité. Ainsi, le système devrait considérer dans le cas des échantillons traités, 15 cas sources candidats dans l'échantillon 1, 15 cas sources dans l'échantillon 2 et 10 cas sources dans l'échantillon 3. Les résultats obtenus en termes de temps de réponse pour chacun des trois échantillons sont exposés dans la figure 8.10. La comparaison des valeurs obtenues pour les échantillons 1 et 2 d'une part et les échantillons 1 et 3 d'autre part montrent que le temps de réponse varie en fonction du nombre de cas dans la communauté concernée et non pas le nombre total de cas de la base de cas.

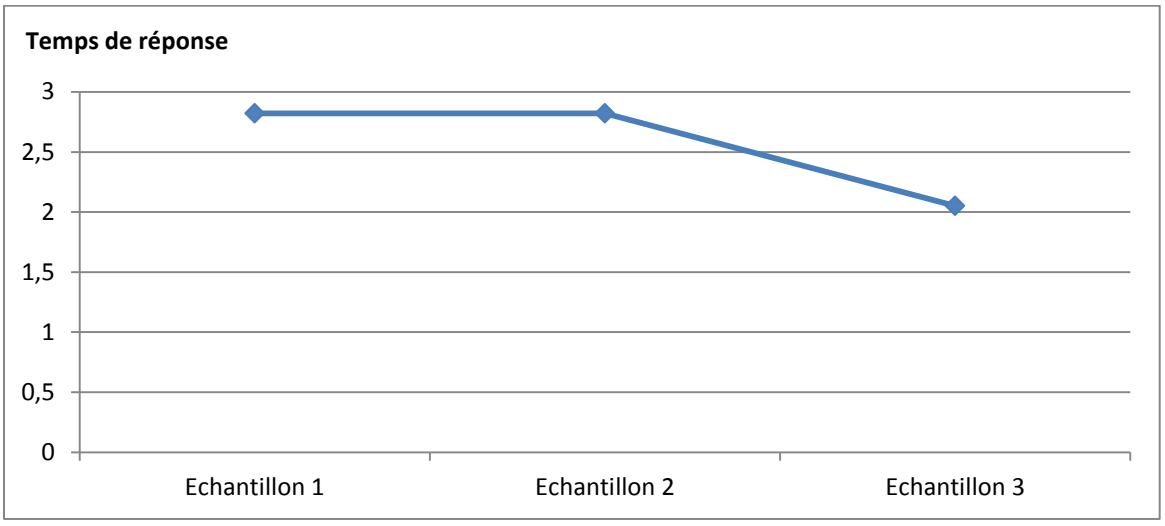


Figure 8.10 : Temps de découverte dans les échantillons testés

8.3.3. Validation des orientations de notre algorithme de remémoration

Notre objectif dans cette section est de montrer la pertinence de nos deux principales orientations à savoir :

- l'utilisation des communautés de services,
- la recherche de similarité en deux temps. Le premier temps consiste à sélectionner les cas similaires fonctionnellement. Ce sous ensemble sert de base au calcul de similarité non fonctionnelle dans un deuxième temps.

Ces deux principes ont pour motivation l'optimisation du temps de remémoration online.

Pour valider ces orientations, nous reprenons l'exemple de notre requête test appliquée dans l'échantillon 1 (cf. tableau 8.9) et nous évaluons la complexité en termes de nombre de comparaisons effectuées lors du calcul de similarité. Pour ce faire, nous considérons trois catégories couvrant les combinaisons des deux critères « utilisation de communauté de services » et « la recherche de similarité en deux temps» et nous calculons le nombre de comparaisons effectuées dans chacune de ces catégories (cf. tableau 8.10).

	Critères adoptés	**Nombre de comparaisons effectuées**
Catégorie 1	-Pas d'utilisation de communautés. -Recherche en un seul temps.	60 cas x 11 descripteurs = 660 Total = **660 comparaisons** **Pas sûr de trouver des solutions après tout le calcul.**
Catégorie 2	-Utilisation de communautés. -Recherche en un seul temps.	Identification de la communauté : 6 27 cas x 12 descripteurs = 324 Total = **330 comparaisons** **Pas sûr de trouver des solutions après tout le calcul.**
Catégorie 3 (CBR4WSD)	-Utilisation de communautés. -Recherche en deux temps.	Identification de la communauté : 6 Verification de découvrabilité : 12 Sélection des cas candidats : 135 15 cas x 6 descripteurs fonctionnels = 90 9 cas x 6 descripteurs non-fonctionnels = 54 Total = **297 comparaisons** **Sûr de trouver des solutions après tout le calcul.**

Tableau 8.10 : Evaluation du nombre de comparaisons effectuées dans les trois catégories.

Les résultats du tableau 8.10 permettent de valider nos choix d'optimisation étant donné que notre système a marqué la plus petite complexité en termes du nombre de comparaisons effectuées lors du calcul de similarité.

8.4. Conclusion

Dans ce chapitre, nous avons présenté les détails d'expérimentation de notre approche. Nous y avons exposé les détails d'expérimentation de cette approche qui a permis de tester les algorithmes conçus moyennant un échantillon d'une base de cas de soixante cas sources du domaine du e-tourisme. L'application de notre prototype à un exemple de requête de réservation de billets d'avion a montré que le système conçu est précis et assez performant, et que le silence obtenu est dû à la manière de calculer les similarités non-fonctionnelles et par la suite l'appariement des propriétés non-fonctionnelles qui n'est pas la meilleure possible. En outre, l'évaluation du temps de réponse du système CBR4WSD dans le cas de trois échantillons de tailles différentes a confirmé que ce temps varie en fonction de la taille de la communauté correspondante à la requête et non pas celle de la base de cas.

D'autre part, l'évaluation de la complexité du système CBR4WSD en termes de nombre de comparaisons effectuées lors du calcul de similarité nous a permis de valider nos choix d'optimisation. Enfin, nous pouvons constater que ces tests ont permis généralement de valider notre approche de découverte sémantique des services Web dans sa globalité.

Conclusion et perspectives

SOMMAIRE

I. Contributions majeures

Dans le présent rapport de thèse, nous avons proposé une approche fondée sur le raisonnement à partir de cas pour la découverte de services Web sémantiques décrits selon les standards W3C WSDL, SAWSDL et WS-Policy. Les objectifs de cette approche varient de la capitalisation sur l'expérience et le traitement sémantique jusqu'à la prise en considération des besoins fonctionnels aussi bien que non-fonctionnels et l'optimisation du temps de découverte. Cette section résume les contributions majeures du travail rapporté dans ce manuscrit.

Une analyse comparative des approches de description des services Web :
Le langage WSDL est un standard W3C adopté dans l'industrie pour la description des services Web implémentant des architectures SOA. Etant donné que ce langage se limite à la description syntaxique des services, de nombreux travaux ont été proposés dans l'objectif d'intégrer la sémantique des services Web, entre autres, nous citons les approches les plus célèbres OWL-S, WSMO et SAWSDL. Une étude comparative détaillée de ces langages nous a permis de dévoiler les mécanismes particuliers de description adoptés par chacun d'eux [El Bitar et al., 2013a] [El Bitar et al., 2013b].

Étant proposé pour étendre WSDL, SAWSDL nécessite peu d'efforts de la part des développeurs habitués à WSDL contrairement à OWL-S et WSMO qui ont été proposées pour concurrencer le standard WSDL en permettant la description de services Web sémantiques. En plus, SAWSDL est ouvert à tout type d'ontologies à l'encontre de OWL-S et WSMO qui exigent l'utilisation d'ontologies spécifiques, à savoir, OWL (pour OWL-S) et WSML (pour WSMO). Notons que ces deux modèles proposent des structures différentes pour la description des services Web et par la suite ont conduit à la proposition d'outils de découverte spécifiques pour l'exploration de chaque structure. Pour le cas de OWL-S qui reste très utilisé dans la littérature, les outils de découverte proposés tel que OWLS-Matcher [Jaeger, 2011], implémentent des algorithmes d'appariement sémantique couvrant uniquement des services annotés suivant l'ontologie OWL-S. Nous notons également que OWL-S et WSMO font recours à WSDL pour apporter les informations d'ordre technique nécessaires pour invoquer un service Web et ne sont pas dédiés à la description des propriétés non-fonctionnelles des services Web, tandis que SAWSDL est un standard d'annotations sémantiques des éléments

du langage de base WSDL qui est un langage extensible. Cette caractéristique de WSDL permet d'y inclure facilement et simplement de nouveaux éléments quand cela est nécessaire dans un but d'obtenir une description assez étoffée des services Web. En plus, ce standard peut être combiné au standard WS-Policy pour permettre de décrire les propriétés non-fonctionnelles des services Web.

Un modèle de description étoffée des services Web étendant le standard WSDL 2.0 et aligné avec les standards SAWSDL et WS-Policy 1.5 [El Bitar et al., 2014a] :
Un modèle de description des services Web expressif, en termes d'intégration de l'aspect sémantique, mais aussi de propriétés aussi bien fonctionnelles que non-fonctionnelles des services Web, demeure fondamental pour le processus de découverte automatique. En plus, ce modèle contribue à assurer l'efficience du processus de découverte en permettant d'améliorer la qualité des résultats par l'intégration des services répondant aux besoins fonctionnels attendus, mais aussi aux exigences et préférences non-fonctionnelles.

En effet, dans notre approche, nous optons pour les standards W3C à savoir WSDL, SAWSDL et WS-Policy en proposant une légère extension de WSDL. Aussi, notre modèle sémantique de description des services Web permet-il, notamment, d'annoter une opération d'un service WSDL par une liste de concepts sémantiques tels que son but (goal), ses postconditions et ses préconditions. Cette extension demeure alignée avec le standard WSDL qui est extensible par définition. Elle consiste à introduire un nouvel attribut que nous appelons « ConceptType » pour spécifier le type d'un concept sémantique (but, précondition, ou postcondition). Nous notons qu'un concept sémantique est un concept d'ontologie référencé au moyen de l'attribut "modelReference" de SAWSDL. Par ailleurs, outre la couverture des aspects fonctionnel et sémantique des services Web, notre modèle couvre également leur aspect non-fonctionnel conformément au standard WS-Policy. Ce dernier permet de définir des politiques de services Web en termes d'exigences, préférences et caractéristiques générales. Il permet, notamment, aux fournisseurs aussi bien que consommateurs de services Web d'exprimer des exigences ou préférences en termes, par exemple, de sécurité ou qualité de service.

Nouvelle approche de découverte de services Web fondée sur le RàPC [El Bitar et al., 2014b] :

La découverte automatique de services Web est généralement fondée sur un mécanisme d'appariement. Cette opération s'avère énormément coûteuse en termes de traitement vu le nombre élevé de services Web disponibles sur Internet. Aussi, toute approche de découverte de services Web devrait-elle être rationnelle en termes de traitement. Autrement, il convient que cette approche soit fondée sur la mutualisation du traitement de manière à ce que les résultats d'un traitement spécifique puissent être partagés entre des requêtes similaires. Or, la majorité des approches de découverte des services Web procèdent à l'appariement de chaque requête reçue avec l'ensemble des services publiés dans un annuaire UDDI.

Dans cette optique de rationalisation du traitement de découverte de services Web, l'approche CBR4WSD que nous avons proposée est fondée sur le retour d'expérience, notamment sur les mécanismes du RàPC (Raisonnement à Partir de Cas). Cela lui permet justement de capitaliser sur l'expérience pour délivrer des solutions inspirées des cas similaires déjà traités, et aussi d'en sélectionner les meilleurs sur la base de l'expérience de leur exécution.

Par ailleurs, outre le fait que notre approche prend en considération les besoins fonctionnels aussi bien que non-fonctionnels, elle adopte une organisation de la base de cas par communautés de services. Le but est de maîtriser la volumétrie des services Web et optimiser le processus de découverte. Cet aspect que nous considérons important n'a pas été pris en considération dans les travaux connexes que nous avons étudiés à l'exception des travaux de [(Osman et al., 2006 a) (Osman et al., 2006 b)] et [(Lajmi et al., 2006a) (Lajmi et al., 2006b)] qui proposent une organisation par domaine métier ou par catégorie de service qui présentent tout de même des limites. L'organisation par communautés de services que nous avons adoptée permet par contre de garantir une remémoration efficace et plus rapide, en réduisant le champ de recherche aux cas sources adéquats et par la suite réduire le temps de découverte. Nous désignons par communauté de services, l'ensemble des services Web qui sont similaires de point de vue fonctionnel indépendamment des exigences ou préférences non-fonctionnelles qu'ils traduisent [Belouadha, 2007].

Il est à noter également que notre approche prévoit deux processus pour la remémoration des cas afin de minimiser le temps de calcul des similarités fondé sur un appariement sémantique.

Un processus online qui couvre cinq couches de traitement : la formalisation, la découverte, la sélection, le test et la mémorisation. Ce processus focalise essentiellement sur l'aspect fonctionnel sans négliger l'aspect non-fonctionnel puisque l'objectif de tout client serait naturellement de trouver une solution qui doit obligatoirement satisfaire son besoin fonctionnel, et qui répond dans la mesure du possible à ses exigences ou préférences non-fonctionnelles.

Un deuxième processus offline couvre une couche transversale assurant l'administration et la configuration du système de découverte, tout en assurant un prétraitement optimisant le temps de remémoration. Notamment, ce processus constitue un processus d'acquisition et de mise à jour de connaissances indispensables pour l'utilisation du système. Déclenché par l'administrateur du système, il permet de construire la base de cas annotés par des concepts d'ontologie de domaine, construire la base de communautés et lancer le calcul de similarités entre les différents concepts sémantiques mis en jeu dans le système en vue de rendre le processus online moins coûteux en termes de temps. Notamment, cela permettrait de minimiser considérablement le temps réservé à l'appariement des cas sources au cas cible en évitant de dupliquer le calcul de similarité entre un concept sémantique du cas cible et un autre qui figure dans plusieurs cas sources.

II. Perspectives

La mise en œuvre de l'approche proposée dans le cadre de ce travail a été réalisée par l'implémentation du système CBR4WSD. La validation des algorithmes que nous avons développés à cet égard a été prouvée par une expérimentation qui a porté sur le domaine du e-tourisme, en particulier les services de réservation de billets d'avion.

Comme perspectives de ce travail, nous envisageons de :

- Etendre l'approche proposée pour permettre la composition de services Web en exploitant toujours le raisonnement à partir de cas et en réutilisant un ensemble de règles de composabilité élaborées dans des travaux précédents de notre équipe de recherche [(Omrana, 2014), (Omrana et al., 2013) (Omrana et al., 2012)].

- Proposer une représentation étoffée des propriétés non-fonctionnelles.
  La représentation que nous avons adoptée est une représentation simple qui couvre un large éventail des propriétés non-fonctionnelles pouvant être exprimées sous forme de contraintes simples. Nous envisageons dans le futur de s'inspirer du profil Marte (Modeling and Analysis of Real-time and Embedded systems) [MARTE, 2011] de description des propriétés non-fonctionnelles des systèmes temps réel et embarqués, en vue de couvrir de façon générique ce type de propriétés.

- Repenser la manière d'apparier les propriétés non-fonctionnelles.
  Le but est d'étudier la possibilité de réaliser des appariements sémantiques centrés sur l'interprétation globale d'une propriété non-fonctionnelle plutôt que sur la sémantique de leurs descripteurs (concept, opérateur, valeur et unité). Cela devrait en fait améliorer la qualité des résultats en permettant de découvrir d'éventuelles solutions possibles dont les descripteurs ne s'apparient pas obligatoirement. A titre d'exemple, une propriété non-fonctionnelle qui traduit un temps de réponse inférieur à 5 ms couvre celle qui exige un temps de réponse inférieur à 10 ms.

Glossaire

WSDL: Web Service Description Language

SAWSDL: Semantic Annotation for Web Service Description Language

OWL-S: Ontology Web Language for Service

WSMO: Web Service Modeling Ontology

UDDI: Universal Description Discovery and Integration

W3C: World Wide Web Consortium

SOA: Service Oriented Architecture

SOAP: Simple Object Access Protocol

RDF: Resource Description Framework

RDQL: RDF Data Query Language

XML: eXtensible Markup Language

QoS: Quality of Service

TFIDF: Term Frequency-Inverse Document Frequency

DSD: Diane Service Description

AASDU: Agent Approach for Service Discovery and Utilization

WSC: Web Service Capabilities

OWLS-M: OWL-S Matchmaker

ALS: Automatic Location of Services

PSWSD: P2P-based Semantic Web Service Discovery

RàPC: Raisonnement à Partir des Cas

CBR: Case Based Reasoning

CASD: Context Aware Service Discovery

LD: Logique de Description

S-CBR: Semantic CBR

WeSCo_CBR: Web Service Composition based on CBR

PF: propriétié fonctionnelle

PNF: propriété non-fonctionnelle

Bibliographie

- [Aamodt et Plaza, 1994] Aamodt, A., Plaza, E., (1994); AICom - Artificial Intelligence Communications, IOS Press, Vol. 7: 1, pp. 39-59.
- [Akkiraju et al., 2005] Akkiraju, R., Farell, J., Miller, J., Nagarajan, M., Schmidt, M., Sheth, A., and Verma, K., Web Service Semantics – WSDL-S, A joint UGA-IBM Technical Note, version 1.0. Technical report, UGA-IBM, April 2005.
- [Aljoumaa et al., 2010] K. Aljoumaa, S. Assar, C. Souveyet, "Publishing intentional services using new annotation for WSDL". In Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services (iiWAS) 8, 10 November 2010, Paris, France.
- [Arabshian et Schulzrinne, 2004], K.. Arabshian, H. Schulzrinne.: Gloserv: Global service discovery architecture. In: MobiQuitous (2004)
- [Armaghan, 2009] Armaghan, N., Contribution à un système de retour d'expérience basé sur le raisonnement à partir de cas conversationnel : application à la gestion des pannes de machine industrielles.(2009)
- [Ayorak et Bener., 2007] Evren Ayorak and Ayse Basar Bener. *Super Peer Web Service Discovery Architecture.* In Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, April 15-20, 2007, Istanbul, Turkey, pages 1360–1364. IEEE, 2007.
- [Baader et al., 2003] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F., editors. The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge, UK, 2003, 555p.
- [Badr et al., 2008] Badr, Y., Abraham, A., Biennier, F., Grosan, C., Enhancing Web Service Selection by User Preferences of Non-functional Features, Next Generation Web Services Practices, NWESP'08. 4th International Conference on pp.60,65, October 2008.
- [Badra et al., 2006] KASIMIR a Medical Decision Support System http://labotalc.loria.fr/~kasimir/

- [Battle et al., 2006] Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., Mellraith, S., McGuinness, D., Su, J., and Tabet, S., Semnatic Web Services Framework (SWSF) Overview. World Wide Web Consortium, Member Submission SUBM-SWSF-20050909, September 2005.
- [Beaubouche, 1994] Beaubouche, N. Anaïs : Raisonnement à partir des cas en résolution de problèmes, Thèse de doctorat, Université Paris 6, Paris, France. (1994).
- [Bello-Tomás et al., 2004] Bello-Tomás, J. J., González-Calero, P. A., & Díaz-Agudo, B. (2004). JColibri: An object-oriented framework for building CBR systems. In Advances in case-based reasoning, 7th European conference, (ECCBR 2004), LNCS (vol. 3155, pp. 32–46). New York: Springer.
- [Belouadha, 2007] Belouadha.F.Z., "Génération dynamique de communautés de services web", Proceedings du Workshop WOTIC'07, Workshop sur les nouvelles Technologies de l'Information et de la Communication, pp. 47-50, Rabat, 5-6 juillet 2007.
- [Bénard et De Loor, 2008] Bénard, R., De Loor, P., " La révision et l'apprentissage de cas pour les simulations temps-réel en réalité virtuelle", In De Loor, P., Bénard, R., (éd) en 16e atelier du Raisonnement à Partir de Cas, avril 2008, Nancy, France.
- [Benatallah et al., 2003] Benatallah, B., Sheng, Q. Z., and Dumas, M. (2003). The Self-Serv Environment for Web Services Composition. IEEE Internet Computing, 7(1).
- [Berners-Lee et al., 2001] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. Scientific American, 284(5) :34–43, 2001.
- [Bernstein et al., 2005] 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.
- [Bernstein et Kiefer., 2005]A. Bernstein and C. Kiefer. iRDQL - Imprecise RDQL Queries Using Similarity Joins. In 3rd International Conference on Knowledge Capture (K-CAP), October 2005.
- [Bertoli, 2004] P. Bertoli, A. Cimatti, and P. Traverso. Interleaving execution and planning for nondeterministic, partially observable domains. In ECAI, pages 657–661, 2004.
- [Branting, 1991] Branting, K. "Exploiting the complementarity of rules and precedents with reciprocity and fairness", In, Proceedings of the Case-Bases Reasoning Workshop 1991, Washington, DC, May 1991, Sponsored by DARPA. Morgan Kaufmann.

- [Bray et al., 2006] Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F., Cowan, J. Extensible Markup Language (XML) 1.1 (Second Edition). W3C Recommendation [en ligne], 2006. Disponible sur : <http://www.w3.org/TR/xml11>.
- [Bruijn J. et al., 2005] Bruijn J. et al., Web Service Modeling Ontology (WSMO). W3C Member Submission, 2005. http://www.w3.org/Submission/WSMO/.
- [Ceausu et Despres, 2006] Ceausu V. et Despres S., Towards a case-based reasoning approach to analyze road accident, Professional Practice in Artificial Intelligence, IFIP International Federation for Information Processing Volume 218, 2006, pp 257-264
- [Chabeb and Tata, 2008] Y. Chabeb and S. Tata,. "Yet Another Semantic Annotation for WSDL (YASA4WSDL)". In IADIS WWW/Internet 2008 Conference pages 462–467, October 2008.
- [Chabeb, 2011] Y. Chabeb. Contributions à la Description et la Découverte de Services Web Sémantiques. Thèse de doctorat de Télécom SudParis dans le cadre de l'école doctorale S&I en co-accréditation avec l'Université d'Evry-Val d'Essonne, Paris, France, Novembre 2011.
- [Charlet et al., 2003] J. Charlet, P. Laublet, and C. Reynaud. Web sémantique Rapport final. Rapport de recherche, Action spécifique 32 CNRS / STIC, FRANCE, Octobre 2003.
- [Chinnici et al, .2007] 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/.
- [Chinnici et al., 2002] Chinnici, R., Moreau J., Ryman, A., and Weerawarana S., Web Services Description Language (WSDL) Version 1.2, 2002. Available at : http://www.w3.org/TR/wsdl12
- [Christos et al., 2006] Christos Makris, Yannis Panagis, Evangelos Sakkopoulos, Athanasios Tsakalidis Efficient and adaptive discovery techniques of Web Services handling large data sets Journal of Systems and Software Volume 79 Issue 4, April 2006, pp. 480–495 Elsevier Science Inc. New York, NY, USA.
- [Chung et al., 1999] Chung L., Nixon B.A., Yu E. and Mylopoulos J., "Non-Functional Requirements in Software Engineering", KluwerAcademic Publishers, Boston Hardbound, October 1999.
- [Claro et al., 2005] 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.

- [Codognet , 1995] Philippe Codognet, Programmation logique avec contraintes : une introduction Technique et Sciences Informatiques, 1995 - info.ucl.ac.be
- [DARPA, 2003] DARPA. DARPA Agent Markup Language. http ://www.daml.org/about.html.
- [De Franco Rosa et De Oliviera, 2008] De Franco Rosa F. et De Oliviera J.M. ; An approach to search Web Services using Ontologies and CBR, The 11th IEEE International Conference on Computational Science and Engineering – Workshops, 2008.
- [Díaz-Agudo et al., 2007] Díaz-Agudo, B., González-Calero, P., Recio-García, J., & Sánchez, A. (2007). Building CBR systems with jCOLIBRI. Journal Science of Computer Programming, 69(1-3), 68–75. (special issue on Experimental Software and Toolkits).
- [Doulkeridis et al., 2005] Doulkeridis C., Loutas N., and Vazirgiannis M.; A system architecture for context-aware service discovery. In International Workshop on Context for Web Services CWS-05, 2005.
- [El Bitar, 2010] I. El Bitar, "CBR: design, implementation and improvement of similarity measures applied to the field of industrial diagnosis", Master Thesis, Lebanese University, Doctoral School of Sciences and Technology, 2010, Beyrouth, Liban.
- [El Bitar et al., 2011a] I. EL Bitar, F.Z. Belouadha, O. Roudies, "Taxonomy and synthesis of Web services querying languages", International Journal of Science and Advanced Technology (ISSN 2221-8386), Volume 1 No 4 June 2011, http://www.ijsat.com
- [El Bitar et al., 2011b] I. EL Bitar, F.Z. Belouadha, O. Roudies "A Logic and Adaptive Approach for Efficient Diagnosis Systems using CBR", International Journal of Computer Applications 39(15):1-5, February 2012. Published by Foundation of Computer Science, New York, USA. BibTeX (ISSN 0975 – 8887)
- [El Bitar et al., 2013 b] El Bitar I., Belouadha F-Z., Roudies O.: Review of Web Services Description approaches, 8th International Conference on Intelligent Systems: Theories and Applications, May 2013, Rabat, Morocco
- [El Bitar et al., 2013a] Omrana H., El Bitar I., Belouadha F-Z., Roudies O.: A Comparative Evaluation of Web Services Description Approaches, 10th International Conference on Information Technology: New Generations ITNG 2013 April 15-17, 2013, Las Vegas, Nevada, USA.
- [El Bitar et al., 2014a] I. EL Bitar, F.Z. Belouadha, O. Roudies, "Towards a semantic description model aligned with W3C standards for WS automatic discovery". The 4th

International Conference on Multimedia Computing and Systems (ICMCS'14) April 14-16 2014, Marrakesh, Morocco

- [El Bitar et al., 2014b] I. EL Bitar, F.Z. Belouadha, O. Roudies, “A CBR based approach for web service automatic discovery”. Journal of Theoretical and Applied Information Technology (ISSN 1992-8645), Volume X No Y 2014, http://www.jatit.org
- [Fan et al., 2005] 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.
- [Farell et Lausen, 2007a] Farrell, J. and H. Lausen (2007). Semantic Annotations for WSDL and XML Schema, W3C recommendation, http://www.w3.org/TR/sawsdl/.
- [Farell et Lausen, 2007b] J. Farrell and H. Lausen. Semantic annotations for WSDL and XML schema. http ://www.w3.org/TR/2007/REC-sawsdl-20070828/, 2007.
- [Fensel et al., 2002] Fensel, D., Bussler, C.,. The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications 1, 2002. Pages : 113-137.
- [Fuchs et al, 1995] Fuchs, B., Chiron, B., et Mille, A. “Decision Helping in Process Supervision: the Padim Project”, Workshop on Practical Development Strategies for industrial Strength Case-Based Reasoning Application, 16th International Conference on Artificial Intelligence, Montréal, Canada.(1995)
- [Fuchs et al., 2006] Fuchs, B., Lieber, J., Mille, A., et Napoli, A. “ Une première formalisation de la phase d’élaboration du raisonnement à partir de cas”, le 14e Atelier de Raisonnement à Partir de Cas, Mars 2006, Besançon, France.
- [Fuchs, 1997] Fuchs, B. , Représentation des connaissances pour le raisonnement à partir de cas, le système ROCADE, Thèse de doctorat, Université Jean Monnet de Saint-Etienne, France.(1997)
- [Fuchs, 2008] Fuchs, B., 2008. Raisonnement à partie de Cas. In Renaud, J., E. Bonjour, B. Chabel-morello, et N. Matta, (éd), Retour et capitalisation d’expérience, outils et démarche. La plainte Saint-Denis : AFNOR, p. 184.
- [Fuglede et Topsoe, 2004] 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.
- [Garcia, 2006] E. Garcia. Cosine similarity and term weight tutorial, 2006.
- [Gebhardt et al, 1997] Gebhardt, F.,Vob, A.,Grather, W. et Schmidt-Beltz, B. Reasoning with Complex Cases, Kluwer Academic, Norwell, MA.(1997)

- [Gusfield, 1997] D. Gusfield. Algorithms on strings, trees, and sequences : computer science and computational biology. Cambridge University Press, New York, NY, USA, 1997.
- [Haj Said, 2004] Haj Said, A. Distances sémantiques pour la comparaison des connaissances objets dans le cadre du raisonnement à partir de cas. (2004)
- [Hammond, 1986] Hammond, K.J. "CHEF: A Model of Case-Based Planning". In Proceeding American Association for Artificial Intelligence, AAAI-86, August 1986. Philadelphia, PA, USA.(1986)
- [Haouchine et al, 2006] Haouchine, K.M., Chebel-Morello, B., et Zerhouni, N. " Methode de suppression de cas pour une maintenance de base de cas", 14e Atelier de Raisonnement à Partir de Cas, Mars 2006, Besançon, France.(2006)
- [Haouchine, 2010] Haouchine, M.K., Remémoration guidée par l'adaptation et maintenance des systèmes de diagnostic industriel par l'approche du raisonnement à partir de cas. Thèse de doctorat, Université de Franche-Comte (2010).
- [Hatzi et al. 2011] Hatzi, O., Vrakas, D., Bassiliades, N., Anagnostopoulos, D., and Vlahavas, I. (2011). The PORSCE II frame-work: Using AI planning for automated semantic web service composition. The Knowledge Engineering ReView, 2011.
- [Henni et al., 2013] Henni, Fouad and Baghdad Atmani. "Applying CBR Over an AI Planner for Dynamic Web Service Composition," International Journal of Information Technology and Web Engineering (IJITWE) 8 (2013): 4, accessed (July 15, 2014), doi:10.4018/ijitwe.2013100103
- [Hinrichs, 1992] Hinrichs T.R. Problem solving in open worlds, Lawrence Erlabaum Associates. Institut FEMTO-ST, UMR CNRS 6174 - UFC / ENSMM / UTBM.(1992)
- [IDEAL, 2011] Intelligent Data Exploration and Analysis Laboratory. Extended Jaccard Similarity. http://www.lans.ece.utexas.edu/ strehl/diss/node56.html (Avril 2011), 2002.
- [Jaeger et al., 2005] 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.
- [Jaeger, 2011] Michael C. Jaeger. OWL-S Matcher.
- [Jena, 2014] http://jena.apache.org/documentation/javadoc/jena/ visited on 21st june 2014
- [Jones, 1972] K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. Journal of Documentation, 28 :11–21, 1972.

- [Kaufer et Klusch, 2006] 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.
- [Keller et al., 2005] 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.
- [Kiefer et al., 2007] C. Kiefer, A. Bernstein, H. J. Lee, M. Klein, and M. Stocker. Semantic process retrieval with iSPARQL. In ESWC, pages 609–623, 2007.
- [Kiefer et Bernstein, 2008] 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.
- [Kiefer, 2009] C. Kiefer. Non-Deductive Reasoning for the Semantic Web and Software Analysis. PhD thesis, University of Zurich, Department of Informatics, Zürich, Switzerland, January 2009.
- [Klein et al., 2005] M. Klein, B. Konig-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.
- [Klein et König -ries, 2004 a] M. Klein and B. Konig-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.
- [Klein et Konig-ries, 2004 b] M. Klein and B. Konig-Ries. Integrating preferences into service requests to automate service usage. In First AKT Workshop on Semantic Web Services, Milton Keynes, UK, December 2004.
- [Klusch et al., 2006] M. Klusch, B. Fries, and K. Sycara. Automated semantic web service discovery with OWLSMX. In Proc. of the fifth international joint conference on Autonomous agents and multiagent systems, pages 915–922, New York, NY, USA, 2006. ACM Press.
- [Klusch et al., 2008] M. Klusch, P. Kapahnke, and F. Kaufer. Evaluation of WSML Service Retrieval with WSMOMX. In IEEE International Conference on Web Services, 2008. ICWS ’08, pages 401–408, Sept. 2008.

- [Klusch et Kapahnke, 2008] 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.
- [Klusch, 2008] Klusch.M., CASCOM Intelligent Service Coordination in the Semantic Web. (2008). Volume: 16, Issue: 2, Publisher: Birkhäuser Basel, Pages: 31-57
- [Kolodner, 1983] Kolodner, J.L. "Maintaining Organization in a Dynamic Long-Term Memory", Cognitive Science, vol. 7, no 4.(1983)
- [Koton, 1989] Koton, P. Using experience in learningand problem solving, Massachusetts Institute of Technology, Laboratory of Computer Science, Ph.D. Thesis, MIT/LCS/TR-441.(1989)
- [Kritikos, 2005] Kritikos, K., Extending OWL for QoS-based Web service description and discovery. In Proceedings of the IBM Ph.D. Symposium at the International Conference on Service-Oriented Computing (ICSOC), A. Hanemann, Ed. 73--78.
- [Küster et al., 2007] 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.
- [Kuster et Konig-Ries, 2007] 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.
- [Küster et Konig-Ries, 2008] U. Küster and B. Konig-Ries. Evaluating semantic web service matchmaking effectiveness based on graded relevance. In Proc. of the 2nd International Workshop SMR on Service Matchmaking and Resource Retrieval in the Semantic Web at the 7th International Semantic Web Conference (ISWC08), Karlsruhe, Germany, October 2008.
- [Lajmi et al. , 2006 a] Lajmi S., Ghedira C., Ghedira K., « How to apply CBR method in web service composition », 2nd International Conference on Signal-Image Technology & Internet based Systems (SITIS'2006), Springer Verlag ed. Hammamet (Tunisie). LNCS series, 2006a.
- [Lajmi et al. , 2006 b] Lajmi S., Ghedira C., Ghedira K., Benslimane D., « Wesco_cbr : How to compose web services via case based reasoning », IEEE International Symposium on Service-Oriented Applications, Integration and Collaboration held with the IEEE

International Conference on e-Business Engineering (ICEBE 2006), Shanghai, China, 2006b

- [Lajmi et al., 2007] Lajmi, S., Ghedira, C., et al. (2007). Une méthode d'apprentissage pour la composition de services web. L'Objet 8(2): pp. 1 - 4.
- [Lamontagne et Lapalme, 2002] Lamontagne, L., Lapalme, G."Raisonnement à Partir de Cas Textuel – état de l'art et perspectives futures", Revue d'Intelligence Artificielle, vol. 16 no 3, Lavoisier, Paris.(2002)
- [Lara et al., 2004] 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.
- [Leake, 1996] Leake, D.B., (1996). CBR in Context: The Present and Future. In, Case Based Reasoning: Experiences, Lessons, & Future Directions, Leake, D.B. (Ed.). AAAI Press / The MIT Press, Menlo Park, Calif., US.
- [Lenz, 1999] Lenz M., Glintschert A., 1999. On Texts, Cases, and Concepts, Proceedings of the 5th Biannual German Conference on Knowledge-Based System (XPS-99), Lecture Notes in Artificial Intelligence 1570, Springer Verlag, pp. 148-156.
- [Letsche T.A. and Berry, 1997] Letsche T.A. and Berry M.W.; Large-scale information retrieval with latent semantic indexing. Information Sciences, 100(1-4):105–137, 1997.
- [Levenshtein, 1996] V. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. Soviet Physics Doklady, 10 :707, 1966. 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.
- [Lieber et Napoli, 1996] Lieber, J., et Napoli, A. "Adaptation of Synthesis Plans in Organic Chemistry", In proceeding of workshop on Adaptation in Case-Based Reasoning, Voss, A., et al. (éd), ECAI-96, Budapest, Hungary. (1996)
- [Lieber, 1997] Lieber, J. Raisonnement à partir de cas et classification hiérarchique. Application à la planification de synthèse en chimie organique. Thèse de doctorat, Université Henri Poincaré Nancy 1, Nancy, France.(1997)
- [Lopez de Mantaras et al., 2005] Lopez de Mantaras R., McSherry D., Bridge D., Leake D., Smyth B., Craw S., Faltings B., Maher M.L., Cox M., Forbus K., Keane M., Aamodt A. et Watson I. Retrieval, Reuse, Revise, and Retention in CBR. Knowledge Engineering Review, pp : 215-240, 2005.

- [Maamar et al., 2009]. Maamar, Z., Subramanian, S., Thiran, P., Benslimane, D., & Bentahar, J. (2009). Communities of Web Services -Concepts, Architecture, Operation, and Deployment. International Journal of EBusiness Research, 5(4), pp. 1-12. Hershey, PA: , IGI Global Publishing.
- [MARTE, 2011] The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems, http://www.omg.org/spec/MARTE/ MARTE 1.1
- [Martin et al., 2004] Martin D. et al., OWL-S: Semantic Markup for Web Services. W3C Submission, 2004. http://www.w3.org/Submission/OWL-S/.
- [McCabe et al., 2003] McCabe H., Newcomer F., Champion E., Ferris M., Orchard C., Booth D., Haas H.; Web services architecture (2003).
- [Mille et al., 1996] Mille, A., Fuchs, B. et Herbeaux, O. " A unifying framework for adaptation in case-based reasoning", Workshop on Adaptation in Case-Based reasoning, European Conference on Artificial Intelligence, ECAI-96, Budapest, Hungary. (1996)
- [Mille, 1995] Mille, A. Raisonnement base sur l'expérience pour coopérer à la prise de décision, un nouveau paradigme en supervision industrielle, Thèse de doctorat, Université Jean Monnet, Saint-Etienne, France. (1995)
- [Mille, 2006] Mille, A. "Tutorial: raisonner à partir de cas: principe, théorisation et ingénierie de la connaissance associée" ; présenté en 14e Atelier du Raisonnement à Partir de Cas, Besançon, France.(2006)
- [myCBR, 2012] http://mycbr-project.net/index.html, last visited on (June 2014).
- [Nadalin et al., 2009] Nadalin, A., M. Goodner, M. Gudgin, A. Barbir, and H. Granqvist (2009). WS-SecurityPolicy 1.3. OASIS standard, http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/ws-securitypolicy.pdf.
- [Nakache et Métais, 2005] NAKACHE D. & MÉTAIS E. (2005). Evaluation : nouvelle approche avec juges. In INFORSID, p. 555–570, Grenoble.
- [OASIS, 2004] WS-Reliability 1.1. OASIS Standard, 2004. Available at http://docs.oasis-pen.org/wsrm/ws-reliability/v1.1/wsrmws_reliability-1.1-spec-os.pdf.
- [Omrana et al., 2012] Hajar Omrana, Fatima-Zahra Belouadha, Ounsa Roudies. A composability Model for efficient Web services' connectivity. IEEE International Conference of Intelligent Networking and Collaborative Systems (INCoS-2012) Romania, September 2012, pages: 483-484

- [Omrana et al., 2013] Hajar Omrana, Fatima-Zahra Belouadha, Ounsa Roudies. A multi aspect rule based model for web services offline composability. Journal of Theoretical and Applied Information Technology. Vol 59 No 2.
- [Omrana, 2014] Omrana, H., Thèse de Doctorat: Vers une composition dynamique des Services Web: une approche de composabilité offline. Ecole Mohammadia d'Ingénieurs, 2014.
- [Osman et al., 2006 a] Osman T., Thakker D., Al-Dabass D.; Semantic-Driven Matchmaking of Web Services Using Case-Based Reasoning. School of Computing and Informatics, Nottingham Trent University, Nottingham, UK, Naval Academy of France, Ecole Navale, BP 600, 29240 Brest, France, 2006
- [Osman et al., 2006 b] Osman T., Thakker D., Al-Dabass D.; S-CBR: Semantic Case Based Reasoner for Web Services Discovery and Matchmaking, School of Computing and Informatics, Nottingham Trent University, Nottingham, UK, Naval Academy of France, Ecole Navale, BP 600, 29240 Brest, France, 2006.
- [Osman et al., 2009] Osman, T., Thakker, D. A., & Al-Dabass, D..(2009), Utilisation of Case-Based Reasoning for Semantic Web Services Composition. International Journal of Intelligent Information Technologies, 5(1)
- [OWL-S Coalition, 2004] 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.
- [Pal et al., 2004] Pal, S.K., Shiu, S.C.K.. Foundation of soft case-based reasoning, Wiley Series on Intelligent Systems IEEE, Albus, J.S., Meystel, A.M., Zadeh, L.A., Series Ediors, ISBN 0471086355, 9780471086352. (2004)
- [Palathingal et Chandra, 2004] Palathingal P. and Chandra S.; Agent approach for service discovery and utilization. In HICSS, 2004.
- [Paolucci et al., 2002] 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.
- [Paolucci et al., 2003] Paolucci, M., Srinivasan, N. Sycara, K., Nishimura, T. Toward a Semantic Choreography of Web Services: From WSDL to DAML-S. In: Procs of the International Conference of Web Services (ICWS03), June 2003, Las Vegas, Nevada, USA. CSREA Press 2003, pp.22-26.

- [Patil et Gopal, 2011] Netra Patil & Arpita Gopal, International Journal of Computer Applications (0975 – 8887) Volume 14– No.1, January 2011 Comparative Study of Mechanisms for Web Service Discovery based on Centralized Approach Focusing on UDDI;
- [Pilioura et Tsalgatidou, 2009] Thomi Pilioura and Aphrodite Tsalgatidou. *Unified publication and discovery of semantic Web services*. ACM Transactions on the Web (TWEB), vol. 3, no. 3, 2009.
- [Reisbeck et Schank, 1989] Reisbeck, C.K, et Schank, R.C. Inside Case-Based Reasoning. Lawrence Erlbaum Associates, Hillsdale, NJ, US.(1989)
- [Rey et al., 2002] Rey C., Hacid M-S., Leger A. and Toumani F.; Dynamic discovery of e-services. Proceedings of BDA'02, 2002, 21-25 octobre 2002 Evry, France.
- [Rey, 2002] Rey C. ; Découverte dynamique de e-services. Symposium on the Effectiveness of Logic in Computer Sciences (ELICS02), March 4-6, 2002, Saarbruecken, Germany.
- [Richter et Wess, 1991] Richter, M.M., et Wess, S. "Similarity, uncertainty and case-based reasoning in PATDEX", In Automated reasoning, essays in honor of Woody Bledsoe, Kluwer, R.S., Boyer (Eds.). (1991)
- [Robeles, 2006]. Robles C, 2006, Management de l'innovation technologique et des connaissances: synergie entre la theorie TRIZ et le Raisonnement a Partir de cas, Ph.D. Thesis, INP, Toulouse.
- [Roland, 2005] Roland C., (2005), L'ingénierie des méthodes : une visite guidée. Revue e-Ti, Numéro 1, 25 octobre 2005.
- [Roman et al., 2006] Roman D., Scicluna J., Fensel D., Polleres A., and Bruijn J,.Ontology-based Choreography of WSMO Services. Wsmo d14 final draft v0.3, DERI, 2006. Available at: http://www.wsmo.org/TR/d14/v0.3/.
- [Salton, 1983] G. Salton. Introduction to Modern Information Retrieval. McGraw-Hill, New York, 1983.
- [Savitha, 2009] Savitha Emani , 2009 A comparative evaluation of semantic webs ervice discovery algorithms and engines, Master od science thesis, Athens, The University of Georgia 2009
- [SAWSDL, 2011] Large Scale Distributed Information Systems. SAWSDL: Semantic Annotations for WSDL. http://lsdis.cs.uga.edu/projects/meteor-s/SAWSDL/ (Avril 2011).

- [Schank, 1982] Schank, RC. “Dynamic memory: a theory of reminding and learning in computers and people”. Cambridge UniversityPress, Cambridge, UK.(1982)
- [Schumacher et al., 2008] M. Schumacher, H. Helin, and H. Schuldt. Semantic Web Service Coordination. Chapter 4, CASCOM: Intelligent Service Coordination in the Semantic Web, Birkhäuser Basel, 2008.
- [Sellami, 2012], Mohamed SELLAMI, 2012, Découverte d’annuaires de services Web dans un environnement distribué Thèse présentée pour l’obtention du grade de Docteur de TELECOM & Management SudParis.
- [Sharma et Kumar, 2011] Sharma, V., et Kumar, M. Web Service Discovery Research: A Study of Existing Approaches, Int. J. on Recent Trends in Engineering & Technology, Vol. 05, No. 01, Mar 2011.
- [Sheth, 2003] Amit P. Sheth. Semantic Web Process Lifecycle: Role of Semantics in Annotation, Discovery, Composition and Orchestration. Invited Talk,Workshop on E-Services and the Semantic Web, at WWW 2003. Available at http://lsdis.cs.uga.edu/lib/presentations/WWW2003-ESSW-invitedTalk-Sheth.pdf.
- [Simpson, 1985] Simpson, R.L. A Computer model of Case-Based Reasoning in Problem Solving: An Investigation in the Domain Dispute Mediation, Technical Report GIT-ICS-85/18, Georgia Institute of Technology, School of Information and Computer Science, Atlanta, USA.(1985)
- [Sivashanmugam et al., 2003] Sivashanmugam K., Verma K., Sheth A.P., and Miller J.A.; Adding semantics to web services standards. In ICWS, pages 395–401, 2003.
- [Smith et al., 2004] M. K. Smith, C. Welty, D. L. McGuinness. OWL Web Ontology Language Guide. http ://www.w3.org/TR/2004/RECowlguide20040210/, Février 2004.
- [Smyth et Keane, 1993] Smyth, B., et Keane, M.T. “ Retrieving Adaptable Cases. The Role of Adaptation Knowledge in Case Retrieval”, Lecture Notes in Artificial Intelligence, vol. 837, Springer-Verlag.(1993)
- [Smyth et Keane, 1995] Smyth, B., et Keane, M.T. “Remembering to Forget: A competence-preserving Deletion Policy for CBR Systems”, In Proceeding of the 14th Joint Conference on Artificial Intelligence, IJCAI-95, Montréal, Canada.
- [Smyth, 1996] Smyth, B. Case-Based Design, Doctoral thesis, Trinity College, Dublin, Ireland.(1996).
- [SPARQL , 2008] SPARQL Query Language for RDF, 2008, http://www.w3.org/TR/rdf-sparql-query/

- [Srinivasan et al., 2004a] Srinivasan N., Paolucci M., Sycara K.; An Efficient Algorithm for OWL-S Based Semantic Search in UDDI, Semantic Web Services and Web Process Composition, Springer-Verlag Berlin Heidelbery, 2004, pp96-100.
- [Srinivasan et al., 2004b] 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.
- [Srinivasan et al., 2006] Srinivasan N., Paolucci M., Sycara K. Semantic Web Service Discovery in the OWL-S IDE, Proceedings of the 39th Annual Hawaii Intel. Conf. on System Sciences (HICSS'06) Track 6, 2006.
- [Stollberg et al., 2007] 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.
- [Sun et al., 2009]. Zhaohao Sun, Jun Han, Dianfu Ma. A Unified CBR Approach for Web Services Discovery, Composition and Recommendation, 2009 International Conference on Machine Learning and Computing IPCSIT vol.3 (2011) © (2011) IACSIT Press, Singapore.
- [Sycara, 1987] Sycara, E.P. Resolving adversarial conflicts: An approach to integrating case-based and analytic methods, Technical Report GIT-ICS-87/26, Georgia Institute of Technology, School of Information and Computer Science, Atlanta, USA.(1987)
- [Teege, 1994], G. Teege, Using description logics in intelligent tutoring systems. Dans Pro c. of the 1994 Description Logic Workshop (DL 94). pp. 7578. (Cité p. 98, 102 et 103)
- [Thompson et al., 2004] Thompson, H.S., Beech, D., Maloney, M., Mendelsohn, N. XML Schema Part 1: Structures Second Edition. W3C Recommendation [en ligne], 2004. Disponible sur : http://www.w3.org/TR/xmlschema-1
- [Toma et al., 2006] Toma, I., Foxvog, D., Jaeger, M.C., Modeling QoS characteristics in WSMO MW4SOC ’06, November 27-December 1, 2006 Melbourne, Australia Copyright 2006 ACM 1-59593-425-1/06/11
- [Vedamuthu et al., 2007] Vedamuthu, A., Orchard, D., Hirsch, F., Hondo, M., Yendluri, P., Boubez, T., and Yalcinalp, U., Web Services policy 1.5 – Framework, W3C recommendation, 2007a. Available at: http://www.w3.org/TR/ws-policy/

- [Vu et al., 2005] Vu L-H., Hauswirth M., and Aberer K. Towards p2p-based semantic web service discovery with qos support. In Business Process Management Workshops, pages 18–31, 2005.
- [Wang et Cao, 2007] Wang L., and Cao J.; Web Services Semantic Searching enhanced by Case Reasoning, 18th International Workshop on Database and Expert Systems Applications, 2007.
- [Warmer et Kleppe, 2003] Jos Warmer Annes Kleppe, The Object Constraint Language: Getting Your Models Ready for MDA, book, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA ©2003 ISBN:0321179366.
- [Watson et Abdullah, 1994]. Watson, I., et Abdullah, S. "Developing case- based reasoning systems: A case study in diagnosing building defects". In, Proc. IEE Colloquium on Case-Based Reasoning: Prospects for Applications, Digest No: 1994/057. (1994)
- [Watson, 1999] Watson, I. "Case-based reasoning is a methodology not a technology" , Knowledge - Based Systems, vol.12, n 5, Elsevier, UK.(1999)
- [Winkler et Thibaudeau, 1991] 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.
- [Zhu et al., 2010] 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.
  http ://owlsm.projects.semwebcentral.org (Avril 2011), 2005.

Annexe A

Extrait du fichier .owl de l'ontologie e-tourism

```
<?xml version="1.0"?>


<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>

<rdf:RDF xmlns="http://e-tourism.deri.at/ont/e-tourism.owl#"
     xml:base="http://e-tourism.deri.at/ont/e-tourism.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <owl:Ontology rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl"/>

    <!--
    ////////////////////////////////////////////////////////////////////////////////
    //
    // Classes
    //
    ////////////////////////////////////////////////////////////////////////////////
     -->
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Activity -->
    <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Activity"/>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#AirlineCompany -->
    <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#AirlineCompany"/>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#ArrivalCity -->
    <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#ArrivalCity">
        <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#City"/>
    </owl:Class>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#ArrivalCountry -->
    <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#ArrivalCountry">
        <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Country"/>
    </owl:Class>
        <!-- http://e-tourism.deri.at/ont/e-tourism.owl#ArrivalDate -->
    <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#ArrivalDate">
        <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Date"/>
    </owl:Class>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#ArrivalTime -->
    <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#ArrivalTime">
        <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Time"/>
    </owl:Class>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Car -->
    <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Car">
        <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Vehicule"/>
    </owl:Class>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#CarRental -->
    <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#CarRental">
        <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Activity"/>
    </owl:Class>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#City -->
    <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#City">
        <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Location"/>
    </owl:Class>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#CloudCover -->
    <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#CloudCover">
        <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#WeatherForecast"/>
    </owl:Class>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Confirmation -->
    <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Confirmation"/>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Country -->
    <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Country">
        <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Location"/>
    </owl:Class>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Currency -->
    <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Currency"/>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#CurrencyExchange -->
    <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#CurrencyExchange">
        <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Activity"/>
    </owl:Class>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Date -->
    <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Date"/>
```

```
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#DepartureCity -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#DepartureCity">
     <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#City"/>
 </owl:Class>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#DepartureCountry -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#DepartureCountry">
     <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Country"/>
 </owl:Class>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#DepartureDate -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#DepartureDate">
     <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Date"/>
 </owl:Class>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#DepartureTime -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#DepartureTime">
     <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Time"/>
 </owl:Class>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#FamilyCar -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#FamilyCar">
     <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Car"/>
 </owl:Class>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#FlightBooking -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#FlightBooking">
     <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Activity"/>
 </owl:Class>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#FlightBookingConfirmation -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#FlightBookingConfirmation">
     <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Confirmation"/>
 </owl:Class>
<!-- http://e-tourism.deri.at/ont/e-tourism.owl#FlightBookingPrice -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#FlightBookingPrice">
    <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Price"/>
 </owl:Class>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#FlightClass -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#FlightClass"/>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#FlightOffers -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#FlightOffers">
     <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Offers"/>
 </owl:Class>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#FlightType -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#FlightType"/>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#HotelBooking -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#HotelBooking">
     <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Activity"/>
 </owl:Class>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#HotelReservationConfirmation -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#HotelReservationConfirmation">
     <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Confirmation"/>
 </owl:Class>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#HotelReservationPrice -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#HotelReservationPrice">
     <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Price"/>
 </owl:Class>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Humidity -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Humidity">
     <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#WeatherForecast"/>
 </owl:Class>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Location -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Location"/>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Meteorology -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Meteorology">
     <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Activity"/>
 </owl:Class>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Motorcycle -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Motorcycle">
     <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Vehicule"/>
 </owl:Class>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Notification -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Notification"/>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Offers -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Offers"/>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Password -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Password"/>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Person -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Person"/>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Precipitation -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Precipitation">
     <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#WeatherForecast"/>
```

```
</owl:Class>
<!-- http://e-tourism.deri.at/ont/e-tourism.owl#Price -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Price"/>
<!-- http://e-tourism.deri.at/ont/e-tourism.owl#RentTime -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#RentTime">
    <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Time"/>
</owl:Class>
<!-- http://e-tourism.deri.at/ont/e-tourism.owl#RentalConfirmation -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#RentalConfirmation">
    <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Confirmation"/>
</owl:Class>
<!-- http://e-tourism.deri.at/ont/e-tourism.owl#RentalDate -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#RentalDate">
    <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Date"/>
</owl:Class>
<!-- http://e-tourism.deri.at/ont/e-tourism.owl#RentalOffers -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#RentalOffers">
    <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Offers"/>
</owl:Class>
<!-- http://e-tourism.deri.at/ont/e-tourism.owl#RentalPrice -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#RentalPrice">
    <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Price"/>
</owl:Class>
<!-- http://e-tourism.deri.at/ont/e-tourism.owl#RestoreDate -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#RestoreDate">
    <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Date"/>
</owl:Class>
<!-- http://e-tourism.deri.at/ont/e-tourism.owl#RestoreTime -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#RestoreTime">
    <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Time"/>
</owl:Class>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#ResultCurrency -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#ResultCurrency">
    <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Currency"/>
</owl:Class>
<!-- http://e-tourism.deri.at/ont/e-tourism.owl#ResultSum -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#ResultSum">
    <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Sum"/>
</owl:Class>
<!-- http://e-tourism.deri.at/ont/e-tourism.owl#Room -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Room"/>
<!-- http://e-tourism.deri.at/ont/e-tourism.owl#RoomOffers -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#RoomOffers">
    <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Offers"/>
</owl:Class>
<!-- http://e-tourism.deri.at/ont/e-tourism.owl#SourceCurrency -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#SourceCurrency">
    <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Currency"/>
</owl:Class>
<!-- http://e-tourism.deri.at/ont/e-tourism.owl#SourceSum -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#SourceSum">
    <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Sum"/>
</owl:Class>
<!-- http://e-tourism.deri.at/ont/e-tourism.owl#SportCar -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#SportCar">
    <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Car"/>
</owl:Class>
<!-- http://e-tourism.deri.at/ont/e-tourism.owl#Sum -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Sum"/>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Temperature -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Temperature">
    <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#WeatherForecast"/>
</owl:Class>
<!-- http://e-tourism.deri.at/ont/e-tourism.owl#Ticket -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Ticket"/>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Time -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Time"/>
<!-- http://e-tourism.deri.at/ont/e-tourism.owl#Truck -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Truck">
    <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Vehicule"/>
</owl:Class>
<!-- http://e-tourism.deri.at/ont/e-tourism.owl#UV-Index -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#UV-Index">
    <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#WeatherForecast"/>
</owl:Class>
<!-- http://e-tourism.deri.at/ont/e-tourism.owl#Vehicule -->
<owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Vehicule"/>
```

```
  <!-- http://e-tourism.deri.at/ont/e-tourism.owl#WeatherForecast -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#WeatherForecast"/>
  <!-- http://e-tourism.deri.at/ont/e-tourism.owl#e-Payment -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#e-Payment">
     <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Activity"/>
 </owl:Class>
  <!-- http://e-tourism.deri.at/ont/e-tourism.owl#e-PaymentConfirmation -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#e-PaymentConfirmation">
     <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Confirmation"/>
 </owl:Class>
  <!-- http://e-tourism.deri.at/ont/e-tourism.owl#e-cardNumber -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#e-cardNumber">
     <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Date"/>
 </owl:Class>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#e-cardPassword -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#e-cardPassword">
     <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Password"/>
 </owl:Class>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#e-cardType -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#e-cardType"/>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#language -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#language"/>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#wind -->
 <owl:Class rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#wind">
     <rdfs:subClassOf rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#WeatherForecast"/>
 </owl:Class>
 <!--
 ///////////////////////////////////////////////////////////////////////////////////
 //
 // Individuals
 //
 ///////////////////////////////////////////////////////////////////////////////////
  -->

<!-- http://e-tourism.deri.at/ont/e-tourism.owl#Air_France -->
 <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Air_France">
     <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#AirlineCompany"/>
 </owl:NamedIndividual>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#American_Express -->
 <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#American_Express">
     <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#e-cardType"/>
 </owl:NamedIndividual>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Arabic -->
 <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Arabic">
     <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#language"/>
 </owl:NamedIndividual>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#BMW -->
 <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#BMW">
     <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#FamilyCar"/>
     <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Motorcycle"/>
     <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#SportCar"/>
 </owl:NamedIndividual>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Business_Class -->
 <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Business_Class">
     <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#FlightClass"/>
 </owl:NamedIndividual>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Economy_Class -->
 <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Economy_Class">
     <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#FlightClass"/>
 </owl:NamedIndividual>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Email_Notification -->
 <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Email_Notification">
     <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Notification"/>
 </owl:NamedIndividual>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#English -->
 <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#English">
     <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#language"/>
 </owl:NamedIndividual>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Euro -->
 <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Euro">
     <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#ResultCurrency"/>
     <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#SourceCurrency"/>
 </owl:NamedIndividual>
 <!-- http://e-tourism.deri.at/ont/e-tourism.owl#First_Class -->
 <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#First_Class">
     <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#FlightClass"/>
 </owl:NamedIndividual>
```

```
   <!-- http://e-tourism.deri.at/ont/e-tourism.owl#French -->
    <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#French">
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#language"/>
    </owl:NamedIndividual>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Honda -->
    <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Honda">
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#FamilyCar"/>
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Motorcycle"/>
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#SportCar"/>
    </owl:NamedIndividual>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Lebanese_Lira -->
    <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Lebanese_Lira">
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#ResultCurrency"/>
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#SourceCurrency"/>
    </owl:NamedIndividual>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#MEA -->
    <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#MEA">
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#AirlineCompany"/>
    </owl:NamedIndividual>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Ma_Dirham -->
    <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Ma_Dirham">
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#ResultCurrency"/>
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#SourceCurrency"/>
    </owl:NamedIndividual>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Master_Card -->
    <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Master_Card">
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#e-cardType"/>
    </owl:NamedIndividual>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Mercedes -->
    <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Mercedes">
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#FamilyCar"/>
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#SportCar"/>
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Truck"/>
    </owl:NamedIndividual>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#OnePersonRoom -->
    <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#OnePersonRoom">
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Room"/>
    </owl:NamedIndividual>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#One_Way -->
    <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#One_Way">
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#FlightType"/>
    </owl:NamedIndividual>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Qatar_Airways -->
    <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Qatar_Airways">
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#AirlineCompany"/>
    </owl:NamedIndividual>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#RAM -->
    <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#RAM">
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#AirlineCompany"/>
    </owl:NamedIndividual>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#SMS_Notification -->
    <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#SMS_Notification">
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Notification"/>
    </owl:NamedIndividual>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Spanish -->
    <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Spanish">
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#language"/>
    </owl:NamedIndividual>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Sweet -->
    <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Sweet">
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Room"/>
    </owl:NamedIndividual>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#ThreePersonsRoom -->
    <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#ThreePersonsRoom">
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Room"/>
    </owl:NamedIndividual>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#TwoPersonsRoom -->
    <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#TwoPersonsRoom">
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#Room"/>
    </owl:NamedIndividual>
    <!-- http://e-tourism.deri.at/ont/e-tourism.owl#Two_Ways -->
    <owl:NamedIndividual rdf:about="http://e-tourism.deri.at/ont/e-tourism.owl#Two_Ways">
        <rdf:type rdf:resource="http://e-tourism.deri.at/ont/e-tourism.owl#FlightType"/>
    </owl:NamedIndividual>
<!-- Generated by the OWL API (version 3.4.2) http://owlapi.sourceforge.net -->
```

Extrait de l'ontologie NFP.owl

```
<?xml version="1.0"?>


<!DOCTYPE Ontology [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>


<Ontology xmlns="http://www.w3.org/2002/07/owl#"
     xml:base="http://www.semanticweb.org/hp/ontologies/2014/3/untitled-ontology-11"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     ontologyIRI="http://www.semanticweb.org/hp/ontologies/2014/3/untitled-ontology-11">
    <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
    <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#"/>
    <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
    <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
    <Declaration>
        <Class IRI="#Price"/>
    </Declaration>
    <Declaration>
        <Class IRI="#ResponseTime"/>
    </Declaration>
    <Declaration>
        <Class IRI="#SecurityLevel"/>
    </Declaration>
    <Declaration>
        <Class IRI="#TextEncoding"/>
    </Declaration>
</Ontology>



<!-- Generated by the OWL API (version 3.4.2) http://owlapi.sourceforge.net -->
```

Annexe B

Classe Cas Source du package CBR4WSD.server.models

```
package com.emi.cbr4wsd.server.models;

import static javax.persistence.GenerationType.IDENTITY;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import org.hibernate.annotations.Cascade;
import org.hibernate.annotations.CascadeType;

@Entity
@Table(name = "CasSource", catalog = "CBR4WSD_DB")
public class CasSource implements java.io.Serializable {

	private static final long serialVersionUID = 1L;
	private Integer id;
	private Communaute communaute;
	private Solution solution;
	private List<CasConcept> casConcepts = new ArrayList<CasConcept>(0);
	private List<Requete> requetes = new ArrayList<Requete>(0);

	public CasSource() {
	}

	public CasSource(Communaute communaute) {
		this.communaute = communaute;
	}

	public CasSource(Communaute communaute, Solution solution,
			List<CasConcept> casConcepts, List<Requete> requetes) {
		this.communaute = communaute;
		this.solution = solution;
		this.casConcepts = casConcepts;
		this.requetes = requetes;
	}

	@Id
	@GeneratedValue(strategy = IDENTITY)
	@Column(name = "id", unique = true, nullable = false)
	public Integer getId() {
		return this.id;
	}

	public void setId(Integer id) {
		this.id = id;
	}

	@ManyToOne(fetch = FetchType.LAZY)
	@JoinColumn(name = "Communaute_id", nullable = false)
	@Cascade(CascadeType.ALL)
	public Communaute getCommunaute() {
		return this.communaute;
	}

	public void setCommunaute(Communaute communaute) {
		this.communaute = communaute;
	}

	@ManyToOne(fetch = FetchType.LAZY)
	@JoinColumn(name = "Solution_id")
	@Cascade(CascadeType.ALL)
	public Solution getSolution() {
		return this.solution;
	}

	public void setSolution(Solution solution) {
		this.solution = solution;
	}
```

```
	@OneToMany(fetch = FetchType.LAZY, mappedBy = "id.casSource")
	@Cascade(CascadeType.ALL)
	public List<CasConcept> getCasConcepts() {
		return this.casConcepts;
	}

	public void setCasConcepts(List<CasConcept> casConcepts) {
		this.casConcepts = casConcepts;
	}

	@OneToMany(fetch = FetchType.LAZY, mappedBy = "casSource")
	@Cascade(CascadeType.ALL)
	public List<Requete> getRequetes() {
		return this.requetes;
	}

	public void setRequetes(List<Requete> requetes) {
		this.requetes = requetes;
	}

	public void addToRequete(Requete requete){
		requete.setCasSource(this);
		requetes.add(requete);
	}

	public void addToCasConcept(CasConcept casConcept){
		casConcept.setCasSource(this);
		this.casConcepts.add(casConcept);
	}

	@Override
	public String toString() {
		return "CasSource [id=" + id + "]";
	}
}
```

Classe Requete du package CBR4WSD.server.models

```
package com.emi.cbr4wsd.server.models;

import java.util.Date;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import static javax.persistence.GenerationType.IDENTITY;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

@Entity
@Table(name = "Requete", catalog = "CBR4WSD_DB")
public class Requete implements java.io.Serializable {

	private static final long serialVersionUID = 1L;
	private Integer id;
	private CasSource casSource;
	private Boolean etatResolution;
	private Date dateLancement;
	private Date tempsReponse;
	private String mailClient;

	public Requete() {
	}

	public Requete(CasSource casSource) {
		this.casSource = casSource;
	}

	public Requete(CasSource casSource, Boolean etatResolution,
```

```
                        Date dateLancement, Date tempsReponse, String mailClient) {
                this.casSource = casSource;
                this.etatResolution = etatResolution;
                this.dateLancement = dateLancement;
                this.tempsReponse = tempsReponse;
                this.mailClient = mailClient;
        }

        @Id
        @GeneratedValue(strategy = IDENTITY)
        @Column(name = "id", unique = true, nullable = false)
        public Integer getId() {
                return this.id;
        }

        public void setId(Integer id) {
                this.id = id;
        }

        @ManyToOne(fetch = FetchType.LAZY)
        @JoinColumn(name = "CasSource_id", nullable = false)
        public CasSource getCasSource() {
                return this.casSource;
        }

        public void setCasSource(CasSource casSource) {
                this.casSource = casSource;
        }

        @Column(name = "etatResolution")
        public Boolean getEtatResolution() {
                return this.etatResolution;
        }

        public void setEtatResolution(Boolean etatResolution) {
                this.etatResolution = etatResolution;
        }

        @Temporal(TemporalType.TIMESTAMP)
        @Column(name = "dateLancement", length = 19)
        public Date getDateLancement() {
                return this.dateLancement;
        }

        public void setDateLancement(Date dateLancement) {
                this.dateLancement = dateLancement;
        }

        @Temporal(TemporalType.TIME)
        @Column(name = "tempsReponse", length = 8)
        public Date getTempsReponse() {
                return this.tempsReponse;
        }

        public void setTempsReponse(Date tempsReponse) {
                this.tempsReponse = tempsReponse;
        }

        @Column(name = "mailClient", length = 45)
        public String getMailClient() {
                return this.mailClient;
        }

        public void setMailClient(String mailClient) {
                this.mailClient = mailClient;
        }

}
```

Classe AdministrateurServiceImpl du package CBR4WSD.server.business

```
package com.emi.cbr4wsd.server.business;

import gwtupload.server.UploadAction;
import gwtupload.server.exceptions.UploadActionException;

import java.io.File;
```

```
import java.util.HashSet;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Set;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import org.apache.commons.fileupload.FileItem;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;
import org.springframework.web.context.request.RequestContextHolder;
import org.springframework.web.context.request.ServletRequestAttributes;

import com.emi.cbr4wsd.server.business.util.JenaHelper;
import com.emi.cbr4wsd.server.dao.Dao;
import com.emi.cbr4wsd.server.models.Administrateur;
import com.emi.cbr4wsd.server.models.CasConcept;
import com.emi.cbr4wsd.server.models.CasSource;
import com.emi.cbr4wsd.server.models.Concept;
import com.emi.cbr4wsd.server.models.FamilleUnite;
import com.emi.cbr4wsd.server.models.Input;
import com.emi.cbr4wsd.server.models.Ontologie;
import com.emi.cbr4wsd.server.models.Output;
import com.emi.cbr4wsd.server.models.ValeurConcept;
import com.emi.cbr4wsd.shared.services.AdministrateurService;
import com.emi.cbr4wsd.shared.services.AuthentificationService;
import com.emi.cbr4wsd.shared.services.OffLineService;
import com.hp.hpl.jena.query.Dataset;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.sdb.SDBFactory;
import com.hp.hpl.jena.util.FileManager;

@Service("administrateurService")
public class AdministrateurServiceImpl extends UploadAction implements
               AdministrateurService, AuthentificationService {

        private static final long serialVersionUID = 1L;
        private String[][] matriceFonctionnelle;

        @Autowired
        @Qualifier("administrateurDao")
        private Dao<Administrateur> adminDao;

        @Autowired
        @Qualifier("casSourceDao")
        private Dao<CasSource> casSourceDao;

        @Autowired
        @Qualifier("ontologieDao")
        private Dao<Ontologie> ontologieDao;

        @Autowired
        @Qualifier("conceptDao")
        private Dao<Concept> conceptDao;

        @Autowired
        @Qualifier("offLineService")
        private OffLineService offLineService;

        @Autowired
        @Qualifier("familleUniteDao")
        private  Dao<FamilleUnite> familleUniteDao;

        private File file;

        public AdministrateurServiceImpl() {

        }

        public int authentification(String login, String password) {
               int id = 0;
               List<Administrateur> admins = adminDao.selectAll();
               for (Administrateur admin : admins) {
                      if (admin.getLoginAdmin().equals(login)
```

```
                                && admin.getPasseAdmin().equals(password)) {
                        id = admin.getId();
                        ServletRequestAttributes           servletRequestAttributes           =
((ServletRequestAttributes) RequestContextHolder
                                        .currentRequestAttributes());
                        HttpSession session = servletRequestAttributes.getRequest()
                                        .getSession(true);
                        session.setAttribute("adminID", id);
                }
        }
        return id;
    }

    public boolean ajoutOntologie(String fileName, Ontologie ontology) {
        String pathToOntologyFile = "ressources/" + fileName;
        Dataset ds = SDBFactory.connectDataset("ressources/sdb-mysql-innodb.ttl");
        Model model = ds.getDefaultModel();
        FileManager.get().readModel(model, pathToOntologyFile);

        String uri = model.getNsPrefixURI("");
        boolean state = true;
        List<Ontologie> ontologies = ontologieDao.selectAll();
        for (Ontologie ontologie : ontologies) {
            if(uri.equals(ontologie.getUri())){
                state = false;
                break;
            }
        }
        if (state) {

        Set<String> conceptsName = JenaHelper.getConceptsNames(model, uri);
        Set<String> valuesName = JenaHelper.getValues(model, uri);
        Ontologie onto = new Ontologie();
        onto.setIntituleOntologie(ontology.getIntituleOntologie());
        onto.setUri(uri);
        onto.setValeurPlugin(ontology.getValeurPlugin());
        onto.setValeurSubsumes(ontology.getValeurSubsumes());
        onto.setSeuilSimilariteFonctionnelle(ontology.getSeuilSimilariteFonctionnelle());

    onto.setSeuilSimilariteNonFonctionnelle(ontology.getSeuilSimilariteNonFonctionnelle());

        LinkedHashMap<String, ValeurConcept> valeursConcept = new LinkedHashMap<>();
        for (String valeur : valuesName) {
            ValeurConcept valeurConcept = new ValeurConcept();
            valeurConcept.setValeur(valeur);
            valeursConcept.put(valeur, valeurConcept);
        }

        for (String conceptName : conceptsName) {
            Concept concept = new Concept();
            concept.setIntituleConcept(conceptName);
            onto.addToConcept(concept);
            Set<String> valuesOfConceptSet = JenaHelper.getValuesOfConcept(
                        model, conceptName, uri);
            if (valuesOfConceptSet.isEmpty()) {
                continue;
            } else {
                for (String key : valuesOfConceptSet) {
                    concept.getValeurConcepts().add(valeursConcept.get(key));
                }
            }
        }
         ontologieDao.insert(onto);
         offLineService.creerMatriceFonctionnelle(conceptsName, uri, model);
        }
        return state;

    }

    public boolean classementConcept(List<Concept> concepts) {
        boolean status = true;
        for (Concept concept : concepts) {
            boolean tmp = conceptDao.update(concept);
            if (!tmp)
                status = tmp;
        }
        return status;
```

```
	}

	public boolean ajoutCas(CasSource casSource) {
		Set<Input> inputs = new HashSet<>();
		Set<Output> outputs = new HashSet<>();
		for (CasConcept casConcept : casSource.getCasConcepts()) {
			if ("input".equals(casConcept.getTypeDescripteur().toLowerCase())) {
				Input in = new Input();
				in.setIntituleInput(casConcept.getConcept()
						.getIntituleConcept());
				inputs.add(in);
			}
			if ("output".equals(casConcept.getTypeDescripteur().toLowerCase())) {
				Output out = new Output();
				out.setIntituleOutput(casConcept.getConcept()
						.getIntituleConcept());
				outputs.add(out);
			}
		}
		for (Output output : outputs) {
			casSource.getCommunaute().addToOutput(output);
		}
		for (Input input : inputs) {
			casSource.getCommunaute().addToInput(input);
		}
		return casSourceDao.insert(casSource);
	}

	public boolean logout() {
		ServletRequestAttributes servletRequestAttributes = ((ServletRequestAttributes)
RequestContextHolder
				.currentRequestAttributes());
		HttpSession session = servletRequestAttributes.getRequest().getSession(
				false);
		if (session == null)
			return false;
		else {
			session.invalidate();
			return true;
		}
	}

	public String executeAction(HttpServletRequest request,
			List<FileItem> sessionFiles) throws UploadActionException {
		System.out.println(">> executeAction invoked !");
		String response = "Received file: ";
		for (FileItem fileItem : sessionFiles) {
			if (!fileItem.isFormField()) {
				try {
					String saveName = fileItem.getName().replaceAll(
							"[\\\\/><\\|\\s\"'{}()\\[\\]]+", "_");
					file = new File("ressources/" + saveName);
					fileItem.write(file);
					System.out.println("File: " + file.getAbsolutePath());
					response += " " + fileItem.getName() + ", size=  "
							+ fileItem.getSize() + ", ContentType= "
							+ fileItem.getContentType()
							+ "Absolute path of File: "
							+ file.getAbsolutePath();
				} catch (Exception e) {
					throw new UploadActionException(e.getMessage());
				}
			}
		}
		try {
			removeSessionFileItems(request);
		} catch (Exception e) {
			throw new UploadActionException(e.getMessage());
		}
		return response;
	}

	public boolean ajouterFamilleUnite(FamilleUnite familleUnite) {
		for(int i=0; i<familleUnite.getUnites().size();i++){
			familleUnite.getUnites().get(i).setFamilleUnite(familleUnite);
		}
		return familleUniteDao.insert(familleUnite);
```

```
        }

         @Override
          public   void   removeItem(HttpServletRequest   request,   String   fieldName)      throws
UploadActionException {
            if (file != null) {
              file.delete();
            }
          }

        public List<FamilleUnite> getAllFamillesUnite() {
                return familleUniteDao.selectAll();
        }

        public List<Concept> getAllConcepts(String ontologie) {
                List<Concept> allConcepts = conceptDao.selectAll();
                List<Concept> conceptOfOntology = new LinkedList<Concept>();
                for (Concept concept : allConcepts) {
                        if
(ontologie.toLowerCase().equals(concept.getOntologie().getIntituleOntologie().toLowerCase())) {
                                conceptOfOntology.add(concept);
                        }
                }
                return conceptOfOntology;
        }

        public boolean updateConceptType(int id, String type){
                Concept conceptToUpdate = conceptDao.select(id);
                conceptToUpdate.setTypeConcept(type);
                return conceptDao.update(conceptToUpdate);
        }

        public Boolean updateConceptFamilleUnit(int id, FamilleUnite familleUnite) {
                Concept conceptToupdate = conceptDao.select(id);
                conceptToupdate.setFamilleUnite(familleUnite);
                return conceptDao.update(conceptToupdate);
        }

        public List<Ontologie> getOntologies() {
                return ontologieDao.selectAll();
        }

}
```

Classe OfflineServiceImpl du package cbr4wsd.server.business

```
package com.emi.cbr4wsd.server.business;

import java.util.Iterator;
import java.util.Set;
import org.springframework.stereotype.Service;
import com.emi.cbr4wsd.server.business.util.JenaHelper;
import com.emi.cbr4wsd.shared.services.OffLineService;
import com.google.gwt.dev.util.collect.HashMap;
import com.hp.hpl.jena.rdf.model.Model;

@Service("offLineService")
public class OffLineServiceImpl implements OffLineService {
        public   HashMap<String,   String[][]>   matricesFonctionnelles   =   new   HashMap<String,
String[][]>();
        public String[][] matriceNonFonctionnelle;
        public String[][] matrice;
        private Iterator<String> itConcepts;
        private int taille;

        public OffLineServiceImpl() {

        }

        @Override
        public  void  creerMatriceFonctionnelle(Set<String>  conceptsName,  String  uri,  Model  model,
String type) {
                taille = conceptsName.size() + 1;
                matrice = new String[taille][taille];
                itConcepts = conceptsName.iterator();

                //Remplissage des entêtes par les concepts
                for(int i=1; i<taille;i++){
```

```
                        if (itConcepts.hasNext()) {
                                String concept = itConcepts.next().toString();
                                matrice[0][i] = concept;
                                matrice[i][0] = concept;
                                matrice[i][i] = "Exact";
                        }
                }

                for (int i = 1; i < taille; i++) {
                        for (int j = 1; j < taille; j++) {
                                if (i != j) {
                                        if (type.equals("nfp")) {
                                                matrice[i][j] = "Fail";
                                        } else {
                                                String    relation    =    JenaHelper.getRelation(model,
matrice[i][0], matrice[0][j], uri);
                                                if (relation.equals("subClassOf")) {
                                                        matrice[i][j] = "Subsumes";
                                                        matrice[j][i] = "Plugin";
                                                        continue;
                                                }
                                                if (relation.equals("Fail") && matrice[i][j] == null &&
matrice[j][i] == null) {
                                                        matrice[i][j] = "Fail";
                                                        matrice[j][i] = "Fail";
                                                }
                                        }
                                }
                        }
                }

                if(type.equals("nfp")){
                        matriceNonFonctionnelle = new String[taille][taille];
                        matriceNonFonctionnelle = matrice;
                } else {
                        matricesFonctionnelles.put(uri, matrice);
                }
        }

        public HashMap<String, String[][]> getMatricesFonctionnelles() {
                return matricesFonctionnelles;
        }

        public String[][] getMatriceNonFonctionnelle() {
                return matriceNonFonctionnelle;
        }

}
```

Classe JenaHelper du package cbr4wsd.server.business.util

```
package com.emi.cbr4wsd.server.business.util;

import java.util.HashSet;
import java.util.Set;

import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.Statement;
import com.hp.hpl.jena.rdf.model.StmtIterator;

public class JenaHelper {

        public JenaHelper() {

        }

        public static Set<String> getConceptsNames(Model model, String uri) {
                Set<String> conceptsName = new HashSet<String>();
                StmtIterator sIter = model.listStatements();
                for (; sIter.hasNext();) {
                        Statement stmt = sIter.nextStatement();
                        String subject = stmt.getSubject().toString();
                        String predicat = stmt.getPredicate().toString();
                        String object = stmt.getObject().toString();
                        if (subject.contains(uri) || predicat.contains(uri) || object.contains(uri)) {
                                String s = subject.substring(subject.indexOf("#") + 1);
                                String p = predicat.substring(predicat.indexOf("#") + 1);
                                String o = object.substring(object.indexOf("#") + 1);
```

```
                        if ("subClassOf".equals(p) && !s.toLowerCase().equals("thing") &&
!o.toLowerCase().equals("thing")
                                        && !s.contains("/") && !o.contains("/")
                                        && !s.contains(":") && !o.contains(":")
                                        && !s.contains("-") && !o.contains("-")) {
                                conceptsName.add(s);
                                conceptsName.add(o);
                        }
                }
        }
        sIter.close();
        return conceptsName;
    }

    public static Set<String> getValues(Model model, String uri) {
        Set<String> values = new HashSet<String>();
        StmtIterator sIter = model.listStatements();
        for (; sIter.hasNext();) {
            Statement stmt = sIter.nextStatement();
            String subject = stmt.getSubject().toString();
            String predicat = stmt.getPredicate().toString();
            if (subject.contains(uri) || predicat.contains(uri)) {
                String s = subject.substring(subject.indexOf("#") + 1);
                String p = predicat.substring(predicat.indexOf("#") + 1);
                if ("type".equals(p)) {
                    values.add(s);
                }
            }
        }
        sIter.close();
        return values;
    }

    public static Set<String> getValuesOfConcept(Model model, String className, String uri){
        Set<String> valuesConcept = new HashSet<String>();
        StmtIterator sIter = model.listStatements();
        for (; sIter.hasNext();) {
            Statement stmt = sIter.nextStatement();
            String subject = stmt.getSubject().toString();
            String predicat = stmt.getPredicate().toString();
            String object = stmt.getObject().toString();
            if (subject.contains(uri) || predicat.contains(uri) || object.contains(uri)) {
                String s = subject.substring(subject.indexOf("#") + 1);
                String p = predicat.substring(predicat.indexOf("#") + 1);
                String o = object.substring(object.indexOf("#") + 1);
                if (className.equals(o)) {
                    if ("type".equals(p) ) {
                        valuesConcept.add(s);
                    }
                }
            }
        }
        sIter.close();
        return valuesConcept;
    }


    public static String getRelation(Model model, String conceptSource, String
conceptDestination, String uri){
        String relationName = "Fail";
        StmtIterator sIter = model.listStatements();
        while(sIter.hasNext()) {
            Statement stmt = sIter.nextStatement();
            String subject = stmt.getSubject().toString();
            String predicat = stmt.getPredicate().toString();
            String object = stmt.getObject().toString();
            if (subject.contains(uri) || predicat.contains(uri) || object.contains(uri)) {
                String s = subject.substring(subject.indexOf("#") + 1);
                String p = predicat.substring(predicat.indexOf("#") + 1);
                String o = object.substring(object.indexOf("#") + 1);

                if (o.equals(conceptSource) && s.equals(conceptDestination)) {
                    relationName = p;
                    break;
                }
            }
        }
```

```
                sIter.close();
                return relationName;
        }
}
```

Extrait de la classe OnLineServiceImpl package com.emi.cbr4wsd.server.business;

```
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map.Entry;
import java.util.Set;

import org.hibernate.Hibernate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import com.emi.cbr4wsd.server.dao.Dao;
import com.emi.cbr4wsd.server.models.CasConcept;
import com.emi.cbr4wsd.server.models.CasSource;
import com.emi.cbr4wsd.server.models.Communaute;
import com.emi.cbr4wsd.server.models.Output;
import com.emi.cbr4wsd.server.models.Solution;
import com.emi.cbr4wsd.shared.services.OffLineService;
import com.emi.cbr4wsd.shared.services.OnLineService;
import org.springframework.stereotype.Service;

@Service("onLineService")
public class OnLineServiceImpl implements OnLineService{
//      //test
//      @Autowired
//      @Qualifier("conceptDao")
//      private Dao<Concept> conceptDao;
//      private AdministrateurService admin = new AdministrateurServiceImpl();


        @Autowired
        private OffLineService offLineService;

        @Autowired
        @Qualifier("communauteDao")
        private Dao<Communaute> communauteDao;

        private String[][] matrice;
        private Communaute communaute;

        public List<Solution> discoverServices(CasSource casSource, String uri){
                List<Solution> liste = new LinkedList<Solution>();
                //recuperer les seuils
                if(verifyDiscovery(casSource, uri)){
                        //effectuer calcul
                        System.out.println("Correct");
                } else {
                        System.out.println("Incorrect");
                        //notifier l'admin
                }
                return liste;
        }

        public void getMatriceSimilarite(String uri){
                Iterator<Entry<String,                String[][]>>                it                =
offLineService.getMatricesFonctionnelles().entrySet().iterator();
                while (it.hasNext()) {
                        Entry<String, String[][]> entree = (Entry<String, String[][]>)it.next();
                 if(entree.getKey().equals(uri)){
                        matrice = entree.getValue();
                        break;
                 }
                 it.remove();
            }
        }

        /* Fonction verification de la decouvrabilite */
```

```
	public boolean verifyDiscovery(CasSource casSource, String uri){
		Set<String> outputsCommunaute = new HashSet<>();
		Set<String> outputsRequete = new HashSet<>();
		for(CasConcept casConcept : casSource.getCasConcepts()){
			if(casConcept.getTypeDescripteur().toLowerCase().equals("output")){
				outputsRequete.add(casConcept.getConcept().getIntituleConcept());
				continue;
			}
		}

		if(verifyCommunity(casSource.getCommunaute().getGoal())){
			getMatriceSimilarite(uri);
			for(Output output : communaute.getOutputs()){
				outputsCommunaute.add(output.getIntituleOutput());
			}
			//if (verifyOutputTotalMatch(outputsRequete, outputsCommunaute)) {
				//createCas();
				int size = selectCases(outputsCommunaute).size();
				if(size>0){
					System.out.println("there are cases");
				} else {
					System.out.println("no one");
				}
//				return true;
//			} else {
//				return false;
//			}
				return true;//à supprimer après
		} else {
			return false;
		}
	}

	/* Verification de l'existence de la communaute */
	public boolean verifyCommunity(String goal){
		boolean state = false;
		List<Communaute> communautes = communauteDao.selectAll();
		for(Communaute communaute : communautes){
			if(goal.equals(communaute.getGoal())){
				this.communaute = communaute;
				state = true;
				break;
			}
		}
		return state;
	}

	/*Fonction de verification de l'existence de tous les outputs d'une requete
	 * dans les outputs de la communaute
	 */
	public boolean verifyOutputTotalMatch(Set<String> outputsRequete, Set<String> outputs){
		int nbMatch = 0;
		for(String output : outputsRequete){
			if(haveOutputMatch(output, outputs)){
				nbMatch++;
			}
		}

		if(nbMatch == outputsRequete.size()){
			return true;
		} else {
			return false;
		}
	}

	/* Fonction verifiant si un output a son correspondant dans la liste des outputs
	 * de la communaute
	 */
	public boolean haveOutputMatch(String output, Set<String> outputs){
		boolean state = false;
		for(String outputCom : outputs){
			if(TabSimMatching(output, outputCom)){
				state = true;
				break;
			}
		}
		return state;
```

```
	}

	/* Verification de la relation entre 2 outputs */
	public boolean TabSimMatching(String source, String destination){
		int indSrc = 0;
		int indDest = 0;

		for(int i=1; i<matrice.length; i++)
		{
			if(matrice[i][0].equals(source)){
				indSrc = i;
			}
			if(matrice[i][0].equals(destination)){
				indDest = i;
			}
		}
		if(matrice[indSrc][indDest].equals("Exact")                                    ||
matrice[indSrc][indDest].equals("Plugin")){
			return true;
		} else {
			return false;
		}
	}

	/* Creation de l'ensemble E contenant les cas sources de la communaute repondants
	 * aux criteres de la decouvrabilite
	 */
	public List<CasSource> selectCases(Set<String> outputs){
		List<CasSource> casSrouces = new LinkedList<CasSource>();
		Set<String> outputsCase = new HashSet<String>();
		System.out.println("ici");
		for(CasSource casSourceCom : this.communaute.getCasSources()){
			System.out.println("centre 1");
			outputsCase.clear();
			//Hibernate.initialize(casSourceCom.getCasConcepts());
			for(CasConcept casConcept : casSourceCom.getCasConcepts()){
				System.out.println("centre 2");
				if(casConcept.getTypeDescripteur().toLowerCase().equals("output")){
					outputsCase.add(casConcept.getConcept().getIntituleConcept());
					System.out.println("concept:
"+casConcept.getConcept().getIntituleConcept());
					continue;
				}
			}
			if(verifyOutputTotalMatch(outputsCase, outputs)){
				casSrouces.add(casSourceCom);
				System.out.println("matched");
			};
		}
		System.out.println("la bas");
		return casSrouces;
	}
}
```

Signature de la communauté « Com1 » :