# Vers une Composition Dynamique des Services Web: une approche de Composabilité Offline

Hajar Omrana

▶ **To cite this version:**

Hajar Omrana. Vers une Composition Dynamique des Services Web: une approche de Composabilité Offline. Web. UNIVERSITÉ MOHAMMED V AGDAL – ECOLE MOHAMMADIA D'INGENIEURS, 2014. Français. NNT : . tel-01134037

**HAL Id: tel-01134037**

**https://theses.hal.science/tel-01134037v1**

Submitted on 21 Mar 2015

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ MOHAMMED V – AGDAL RABAT

**ECOLE MOHAMMADIA D'INGENIEURS**

Centre d'Etudes Doctorales

Sciences et Techniques pour l'Ingénieur

## Thèse de Doctorat

# Vers une composition dynamique des Services Web: une approche de composabilité offline

Présentée et soutenue publiquement le 20 Janvier 2014
Par

**Hajar OMRANA**

Pour obtenir le grade de

**Docteur en Sciences et Techniques pour l'Ingénieur**

**Devant le jury composé de**

| | | |
|---|---|---|
| Pr. Dalila CHIADMI | Président | EMI |
| Pr. Ounsa ROUDIES | Directeur de thèse | EMI |
| Pr. Fatima-Zahra BELOUADHA | Co-directeur de thèse | EMI |
| Pr. Omar EL BEQQALI | Rapporteur | FSDM |
| Pr. Laila BENHLIMA | Rapporteur | EMI |
| Pr. Mohammed KHALIDI IDRISSI | Rapporteur | EMI |
| Pr. Mounia FREDJ | Examinateur | ENSIAS |
| Pr. Najib TOUNSI | Examinateur | EMI |

# Résumé

La composition dynamique reste un des objectifs clé de la technologie des services Web. Alors qu'ils sont conçus pour être agrégés et collaborer ensemble, les services Web peuvent en même temps être mis en œuvre indépendamment et utiliser des normes ou modèles différents. Par conséquent, leur processus de composition dynamique s'avère complexe et coûteux en temps. Vérifier la possibilité de connecter des services (la composabilité) en amont permet d'accroître considérablement, l'efficacité et l'exactitude de ce processus. Dans la plupart des travaux sur la composition des services Web, la composabilité de deux services ou opérations de services se limite souvent aux processus d'appariement sémantique et/ou syntaxique des entrées et sorties des services ou opérations concernés, et ne traite pas la cohérence des propriétés techniques, non-fonctionnelles, structurelles ou contextuelles des deux services.

Dans le but d'aboutir à des plans de composition efficaces et d'optimiser le temps de composition dynamique, notre recherche propose une approche de composabilité offline. Cette approche consiste à identifier l'ensemble de services composables à différents niveaux, en amont du processus de construction des plans de composition devant être effectué de façon dynamique.

Elle définit d'abord un modèle de description de services Web multi-aspects qui s'aligne avec les spécifications W3C, à savoir, WSDL 2.0, SAWSDL et WS-Policy 1.5. Ce modèle intègre les propriétés descriptives prévues par ces trois standards et les enrichit par de nouvelles propriétés, dans le but de capturer le maximum d'informations sur un service Web, tout en restant conforme aux standards. Sur la base de ce modèle descriptif, notre approche identifie les propriétés descriptives impliquées dans la composabilité offline des services et définit six règles multi-aspects qui exploitent ces informations pour traiter les aspects de composabilité de deux opérations de services Web : fonctionnel, non-fonctionnel, contextuel, orienté données et technique. Nous définissons aussi une démarche globale de vérification automatique des aspects de composabilité offline de deux opérations. Cette démarche comprend trois principales phases : (i) l'extraction et le stockage des informations de composabilité à partir des fichiers SAWSDL, WSDL 2.0 et WS-Policy, (ii) la vérification automatique de la composabilité offline de deux opérations par l'algorithme que nous avons développé à cette fin, et enfin (iii) la traçabilité des résultats de composabilité des opérations. La faisabilité de cette démarche a été démontrée par le dispositif de composabilité offline que nous avons réalisé.

***Mots clés** : Composabilité offline, composition dynamique, WSDL2.0, SAWSDL, WS-Policy 1.5, description de services Web.*

# Abstract

Dynamic composition remains one of the key aims of Web services technology. While they are designed to be aggregated and work together, Web services can be simultaneously implemented independently and use different standards or models. Therefore, their dynamic composition process is complex and time-consuming. Checking the possibility of connecting services (composability) enhances significantly this process's efficiency and accuracy. In most works relating to Web services composition, the composability of two services or service operations is often limited to the process of semantic and/or syntactic matching of the inputs and outputs of the services or the operations concerned, and does not deal with the consistency of technical, non-functional, structural or contextual properties of the two services.

In order to achieve efficient composition plans and optimize dynamic composition time, this work suggests an offline composability approach. The latter aims at identifying all the services which can be composed at different levels, as a preliminary, the process of construction of composition plans to be performed dynamically.

It defines beforehand a multi-aspect Web service description model, which is aligned with the W3C specifications, namely, WSDL 2.0, SAWSDL and WS-Policy 1.5. This model incorporates the descriptive properties provided by these three standards and enriches them with new properties, in order to get as much information about a Web service, while remaining compliant with the standards. Based on this descriptive model, our approach identifies the descriptive properties involved in the offline composability of services and defines a set of multi-aspect rules which use this information in order to deal with the aspects of composability of two Web service operations: functional, non-functional, contextual, data-driven and technical. This approach defines also the overall method of automatic checking of the aspects of offline composability of two operations. This method consists of three main phases: (i) extraction and storage of composability information from SAWSDL, WSDL 2.0 and WS-Policy files, (ii) automatic checking of the offline composability of two operations using an algorithm that we have developed for this purpose, and (iii) traceability of the operation composability results. An experimentation part is presented within the framework of this research so as to clarify the use of the developed offline composability mechanisms.

***Key words** : Offline Composability, dynamic composition, WSDL2.0, SAWSDL, WS-Policy 1.5, Web services description.*

# Liste des publications

- ***Publications dans un chapitre du livre***

  - ✓ Fatima-Zahra Belouadha, Hajar Omrana et Ounsa Roudiès (2012). Web services-enhanced agile modeling and integrating business processes. E-Business - Applications and Global Acceptance, ISBN 978-953-51-0081-2; Edited by: Princely Ifinedo; Publisher: InTech, February 2012, pages: 73-98

- ***Publications dans des revues indexées***

  - ✓ Hajar Omrana, Fatima-Zahra Belouadha, et Ounsa Roudiès (2013). MARTE Profile-Based MDA Approach for Semantic NFP-Aware Web Services. European Journal of Scientific Research, Volume 94, issue 4, 2013

  - ✓ Hajar Omrana, Fatima-Zahra Belouadha, et Ounsa Roudiès (2013). A MULTI-ASPECT RULE BASED MODEL FOR WEB SERVICES OFFLINE COMPOSABILITY. Journal of Theoretical and Applied Information Technology, (Accepted Paper).

  - ✓ Hajar Omrana, Fatima-Zahra Belouadha, et Ounsa Roudiès (2012). Template–based matching algorithm for dynamic web services discovery. International Journal of Information and Communication Technology, **Inderscience Publishers**, Volume 4, Issue 2/3/4. Pages: 198-209

  - ✓ Hajar Omrana, Fatima-Zahra Belouadha, et Ounsa Roudiès (2012). Design for Distributed Moroccan Hospital Pharmacy Information Environment with Service Oriented Architecture. International Journal on Informatics for Development, Volume 1, Issue 1, 2012, pages: 26-31

- ✓ Hajar Omrana, Fatima-Zahra Belouadha, et Ounsa Roudiès (2011). UML Based Profiles for Policy-aware Web services. The International Journal of Reasoning-based Intelligent Systems (IJRIS), Volume 3, Issue ¾, **Inderscience Publishers**, 2011, pages: 217-225

- ✓ Fatima-Zahra Belouadha, Hajar Omrana et Ounsa Roudiès (2010). A model-driven approach for composing SAWSDL semantic Web services. IJCSI international journal, Volume 7, issue 2, 2010, pages : 7-15, ISSN (Online) : 1694-0784, ISSN (Print) : 1694-0814, http://www.ijcsi.org/papers/7-2-1-7-15.pdf.

- ✓ Fatima-Zahra Belouadha, Hajar Omrana et Ounsa Roudiès (2010). A MDA approach for defining WS-Policy semantic non-functional properties. International Journal of Engineering Science and Technology (IJEST), Volume 2, Issue 6, 2010, pages: 2164-2171, ISSN: 0975-5462, http://www.ijest.info/docs/IJEST10-02-06-115.pdf.

- ***Conférences internationales indexées avec publication d'actes***

  - ✓ Hajar Omrana, Ibrahim El Bitar, Fatima-Zahra Belouadha et Ounsa Roudiés (2013). A Comparative Evaluation of Web Services Description Approaches. ITNG '13 Proceedings of the IEEE 10th International Conference on Information Technology: New Generations, Las Vegas, Nevada, USA, April 2013, pages: 60-64

  - ✓ Hajar Omrana, Fatima-Zahra Belouadha and Ounsa Roudiés (2012). A composability Model for efficient Web services's connectivity. IEEE International Conference of Intelligent Networking and Collaborative Systems (INCoS-2012) Romania, September 2012, pages: 483-484.

  - ✓ Hajar Omrana, Fatima-Zahra Belouadha et Ounsa Roudiès (2011). Dynamic Web Services Discovery Approach Based on Functional and non-Functional Requirements, International Conference on Informatics for Development

(ICID'11), Indonesia. **(Selected Best Paper, Invited for an Special Issue of The International Journal of Information and Communication Technology).**

- ✓ Hajar Omrana, Safae Nassiri, Fatima-Zahra Belouadha and Ounsa Roudiés (2011). Towards a Web services-based e-health architecture: Moroccan Hospital Pharmacy Information System Case , International Conference on Informatics for Development (ICID), Indonesia, 2011. **(Invited for an Special Issue of The International Journal on Informatics for Development )**

- ✓ Hajar Omrana, Fatima-Zahra Belouadha et Ounsa Roudiès (2010). A MDA approach for describing Web services policies. The Third International Conference on Web and Information Technologies ICWIT (Marrakech), 2010, pages: 449-460, ISBN: 978-9954- 9083-0-3, **(Selected Best Paper, Invited for an Special Issue of The International Journal of Reasoning-based Intelligent Systems )**

- ***Conférences nationales***

  - ✓ Hajar Omrana, Fatima-Zahra Belouadha et Ounsa Roudiès (2010). Méta-modélisation des propriétés des services Web alignée avec les standards W3C. JDTIC 2011.

  - ✓ Fatima-Zahra Belouadha, Hajar Omrana et Ounsa Roudiès (2012). Les processus métier : quel langage de modélisation?. JODIC 2012

# Table de matières

## Chapitre 1 - Introduction


***1.1. Contexte de la thèse*** ..... ***13***

***1.2. Motivation et problématique*** ..... ***14***

***1.3. Fondements de notre approche*** ..... ***16***

***1.4. Contributions de la thèse*** ..... ***17***

***1.5. Organisation de la thèse*** ..... ***17***


## Chapitre 2 - Etat de l'Art de la description de services Web


***2.1. Introduction*** ..... ***21***

***2.2. Propriétés des services Web*** ..... ***22***

2.2.1. Propriétés fonctionnelles ..... 23

2.2.2. Propriétés non-fonctionnelles ..... 24

2.2.3. Propriétés contextuelles ..... 24

2.2.4. Propriétés orientées données ..... 24

2.2.5. Propriétés techniques ..... 25

2.2.6. Propriétés comportementales ..... 25

2.2.7. Synthèse ..... 26

***2.3. Classification des approches de description des services Web*** ..... ***26***

2.3.1. Niveaux de description syntaxique et sémantique ..... 27

2.3.2. Approches à base de langages syntaxiques ..... 28

2.3.3. Approches à base de modèles sémantiques ..... 28

***2.4. Standards W3C : WSDL, SAWSDL et WS-Policy*** ..... ***28***

2.4.1. Web Services Description Language (WSDL) ..... 30

2.4.2. Semantic Annotations for WSDL (SAWSDL) ..... 32

2.4.3. Web Services Policy (WS-Policy) ..... 34

***2.5. Modèles sémantiques : OWL-S et WSMO*** ..... ***36***

2.5.1. Ontology Web Language for Service (OWL-S) ..... 37

2.5.2. Web Services Modelling Ontology (WSMO) ..... 39
***2.6. Analyse des approches de description des services Web ..... 42***
2.6.1. Couverture de la description fonctionnelle ..... 42
2.6.1.1. Cas des langages WSDL2.0 et SAWSDL ..... 42
2.6.1.2. Cas de l'ontologie OWL-S ..... 43
2.6.1.3. Cas de l'ontologie WSMO ..... 43
2.6.1.4. Discussion ..... 44
2.6.2. Couverture de la description de données ..... 46
2.6.2.1. Cas de WSDL/SAWSDL ..... 46
2.6.2.2. Cas de OWL-S et WSMO ..... 47
2.6.2.3. Discussion ..... 48
2.6.3. Couverture de la description non-fonctionnelle ..... 48
2.6.3.1. Cas de WSDL/WS-Policy ..... 49
2.6.3.2. Cas de OWL-S et WSMO ..... 50
2.6.3.3. Discussion ..... 50
2.6.4. Couverture de la description technique ..... 51
2.6.4.1. Cas de WSDL ..... 51
2.6.4.2. Cas de OWL-S et WSMO ..... 52
2.6.4.3. Discussion ..... 52
***2.7. Synthèse ..... 52***

# Chapitre 3 - Etat de l'Art de la composition de services Web

***3.1. Introduction ..... 56***
***3.2. Paradigme de composition des services Web ..... 57***
3.2.1 Définitions et cycle de vie de la composition des services Web ..... 57
3.2.2 Propriétés comportementales des services Web composites ..... 60
3.2.1.1. Orchestration ..... 60
3.2.1.2. Chorégraphie ..... 61
3.2.3 Composition statique Vs. composition dynamique ..... 63
***3.3. Découverte des services Web ..... 65***
3.3.1 Spécification de la requête utilisateur ..... 66
3.3.2 Localisation des services Web ..... 67
3.3.3 Appariement des requêtes et des services Web ..... 68

3.3.4 Synthèse ..... 70
***3.4. Construction des plans de composition des services Web ..... 71***
3.4.1 Approches par Workflows ..... 72
3.4.2. Approches par techniques de planification ..... 75
3.4.3. Approches par graphes de dépendances ..... 80
3.4.4. Synthèse ..... 82
***3.5. Discussion ..... 82***

## Chapitre 4 - Modèle UML de description des services Web

***4.1. Introduction ..... 87***
***4.2. Description des propriétés fonctionnelles des services Web ..... 88***
4.2.1. Modèle de base pour la description abstraite des services Web ..... 88
4.2.2. Modèle d'annotation sémantique des propriétés fonctionnelles ..... 90
4.2.3. Description des propriétés contextuelles et conditions des opérations : limites des standards W3C ..... 92
4.2.4. Modèle proposé de description des propriétés contextuelles des paramètres ..... 94
4.2.4.1. Aperçu des solutions pour la description des propriétés contextuelles ..... 94
4.2.4.2. Solution proposée pour la description des propriétés contextuelles des paramètres des services Web ..... 95
4.2.5. Modèle proposé de description des conditions d'opérations ..... 99
4.2.6. Modèle global de la description fonctionnelle des services Web ..... 101
***4.3. Description des données sémantiques ..... 103***
4.3.1. Mécanisme d'échange de données entre les services Web ..... 103
4.3.2. Modèle de description des données sémantiques ..... 103
***4.4. Description de l'aspect technique des services Web ..... 105***
4.4.1. Mécanisme d'accès aux services Web WSDL ..... 105
4.4.2. Modèle de description des propriétés techniques des services Web ..... 105
***4.5. Description de l'aspect non-fonctionnel des services Web ..... 106***
4.5.1. Composants et exemple d'une politique de services Web ..... 107
4.5.2. Modèle de description des politiques de service Web ..... 109
4.5.3. Limites de WS-Policy 1.5 ..... 110
4.5.4. Mécanisme d'annotation sémantique et contextuelle d'une assertion ..... 112
***4.6. Synthèse ..... 114***

## Chapitre 5 - Approche de composabilité offline des services Web


***5.1. Introduction ... 117***

***5.2 Composabilité des services Web : Positionnement et enjeux ... 118***

5.2.1 Composabilité Vs Composition ... 119

5.2.2 Limites de l'approche classique de composabilité ... 119

***5.3. Fondements de notre approche de composabilité des services Web ... 120***

5.3.1 Champs et cas de composabilité ... 120

5.3.2 Composabilité offline et online ... 123

5.3.3 Synthèse ... 125

***5.4. Concepts et mécanismes de la composabilité offline des services Web ... 126***

5.4.1 Hypothèses de base ... 126

5.4.2 Terminologie et définitions ... 127

5.4.4 Synthèse ... 134

***5.5. Règles de composabilité offline ... 135***

5.5.1 Règles de composabilité « stricte » et « souple » ... 135

5.5.2 Règle de composabilité fonctionnelle ... 136

5.5.3 Règle de composabilité non-fonctionnelle ... 139

5.5.4 Règles de composabilité contextuelle ... 142

5.5.5 Règle de composabilité orientée données ... 144

5.5.6 Règle de composabilité technique ... 146

***5.6 Synthèse et discussion ... 147***


## Chapitre 6 - Dispositif de composabilité des services Web


***6.1. Introduction ... 151***

***6.2. Conception du dispositif ... 151***

6.2.1. Démarche globale ... 151

6.2.2. Base de données de composabilité ... 152

6.2.3. Environnement technique ... 155

***6.3. Algorithme de composabilité offline ... 156***

6.3.1. Fonction de composabilité fonctionnelle ... 156

6.3.2. Fonction de composabilité contextuelle (niveau fonctionnel) ... 157

6.3.3. Fonction de composabilité orientée données ... 158

6.3.4. Fonction de vérification de satisfaction des politiques (composition non-fonctionnelle) ..... 159
6.3.5. Fonction de composabilité technique ..... 162
6.3.6. Vue globale de l'algorithme de composabilité offline ..... 162
***6.4. Expérimentation ..... 167***
6.4.1. Présentation des opérations candidates ..... 167
6.4.2. Présentation des scénarios de test ..... 169
6.4.3. Résultat du scénario 1 ..... 170
6.4.4. Résultat du scénario 2 ..... 172
6.4.5. Résultats du scénario 3 ..... 172
6.4.6. Contenu de la table *ComposabilityTrace* ..... 173
***6.5. Synthèse ..... 174***

# Chapitre 7 - Conclusion

***7.1. Contributions majeures ..... 177***
***7.2. Perspectives ..... 180***

***Bibliographie ..... 182***

# Chapitre 1

# Introduction

## Sommaire

*1.1. Contexte de la thèse ..... 13*

*1.2. Motivation et problématique ..... 14*

*1.3. Fondements de notre approche ..... 16*

*1.4. Contributions de la thèse ..... 17*

*1.5. Organisation de la thèse ..... 17*

## 1.1. Contexte de la thèse

De plus en plus, la compétitivité de l'entreprise dépend de sa capacité à générer, à traiter et à analyser rapidement et efficacement l'information pour prendre des décisions pertinentes et acquérir un avantage concurrentiel, que ce soit dans sa relation avec ses clients, ses fournisseurs, ses collaborateurs ou ses partenaires. L'entreprise est incessamment exposée à des défis concurrentiels auxquels elle doit continuellement réagir. En effet, elle doit répondre aux enjeux concurrentiels du marché en améliorant la qualité de son offre, en proposant de nouvelles fonctionnalités pour étendre son activité commerciale, et en s'ouvrant sur les systèmes d'information de ses partenaires pour répondre au besoin du partage instantané de l'information, et cela dans des délais optimums.

Grâce au développement accéléré des technologies de l'information (TI) et l'avènement des services Web, l'Internet se transforme d'un simple vecteur d'échange de données, en une plate-forme de composants auto-descriptifs, modulaires, facilement intégrables et faiblement couplés. Récemment, les services Web ont émergé comme l'instanciation *de-facto* de l'approche SOA (*Service Oriented Architecture*) [Oppong et al., 2010], permettant ainsi à l'entreprise de se libérer de sa position fermée et d'explorer de nouvelles pistes de collaboration avec d'autres entreprises, afin de créer de nouvelles fonctionnalités, et ce en réutilisant et en agrégeant ses propres services Web existants ou en utilisant les services Web de ses partenaires.

Les services Web individuels restant limités par leurs capacités, l'entreprise est amenée à composer un ensemble de services afin de créer des services plus complexes. On parle ainsi de la composition des services Web [Alonso et al., 2004]. Cette composition se présente comme un paradigme fondamental de la technologie des services Web. Elle permet de résoudre des problèmes complexes en combinant des services de base disponibles pour satisfaire un but initial. Ce paradigme demeure l'un des axes de recherche les plus actifs des services Web, durant ces dix dernières années, vue la complexité du processus de composition et l'évolution rapide des normes et des standards de cette technologie.

La composition des services Web est un processus complexe qui fait intervenir plusieurs activités telles que la découverte, la composabilité, la sélection ou encore

l'exécution des services Web. La composabilité, notion peu investie dans la littérature, permet de vérifier si un ou plusieurs services, voire opérations de services, peuvent interagir entre eux dans le but de construire des plans de composition. Elle constitue en fait, un processus crucial dont dépend l'efficacité des plans de composition, et par la suite l'efficacité du processus global de composition des services Web. Pour être efficient, le processus de composabilité de services Web doit en pratique vérifier la cohérence de propriétés relatives aux différents aspects des services Web : fonctionnel, non-fonctionnel, technique, structurel et contextuel. L'étude en amont de la composabilité des services Web permet de réduire la liste des services candidats pour la composition dynamique. Cet avantage est d'autant plus appréciable avec l'offre croissante de services Web publiés.

C'est enfin dans ce contexte que s'inscrit ce travail. Il propose une approche alignée avec les standards W3C (*World Wide Web Consortium*) pour la vérification de la composabilité des services Web en amont (désignée dans ce rapport par la notion de composabilité offline). L'objectif étant de permettre d'identifier les opérations de services Web pouvant être interconnectées, au préalable des requêtes reçues par un processus de composition dynamique. Cela devrait par la suite contribuer à aboutir à des plans de composition efficients et exécutables.

## 1.2. Motivation et problématique

Composer deux services Web S1 et S2 revient à connecter une opération de S1 avec une autre opération de S2. Dans la plupart des travaux de composition étudiés, la connexion de ces deux opérations est fondée sur la vérification de la couverture des entrées de l'une par les sorties de l'autre, en utilisant des appariements sémantique et/ ou syntaxique. Ces travaux se limitent souvent aux processus d'appariement sémantique ou syntaxique des entrées et des sorties des services ou des opérations concernés, et ne traitent pas l'ensemble des informations techniques, non-fonctionnelles, structurelles ou contextuelles incorporées dans un fichier WSDL (*Web Services Description Language*). Les propriétés non-fonctionnelles, à titre d'exemple, lorsqu'elles sont décrites, s'avèrent souvent exploitées dans la phase de sélection du meilleur plan de composition. Or, en réalité le critère de composabilité des paramètres des services Web demeure seul insuffisant pour produire des plans de composition efficaces. D'autres propriétés

descriptives des services Web doivent être prises en considération pour pouvoir connecter des services, souvent développés indépendamment et par divers fournisseurs, et avoir des plans de composition efficaces et exécutables. Notamment, des propriétés techniques comme les protocoles de communication, qui peuvent constituer un obstacle lors de l'exécution des plans de composition obtenus si leur compatibilité n'a pas été vérifiée.

D'autre part, plusieurs solutions de composition proposées dans la littérature restent expressivement liées aux propriétés descriptives fournies par le modèle sémantique de services adopté (par exemple, OWL-S « *Ontology Web Language for Service* », WSMO « *Web Services Modelling Ontology* »). En d'autres termes, et vue les hétérogénéités qui peuvent exister entre les propriétés descriptives de différents modèles sémantiques, la solution de composition proposée se limite ainsi aux services décrits par le modèle sémantique utilisé. Cela impacte certes le champ de services candidats couverts par la solution, mais aussi l'efficacité et la longévité de la solution.

Par ailleurs, cette catégorie de solutions optant pour des modèles sémantiques, renvoie aux fichiers WSDL des services Web pour toute information technique. Toutefois, elle ne traite pas cette information technique au moment de la vérification de la composabilité des services. De ce fait, les plans de composition qu'elle génère demeurent des plans abstraits dont la composabilité réelle des services n'a pas été vérifiée.

Par conséquent, les deux limites majeures relevées dans les traitements existants de composabilté sont :

- ✓ l'hétérogénéité des modèles adoptés pour la description des services Web.
- ✓ l'évaluation de la composabilité qui demeure à un niveau abstrait. Elle met l'accent uniquement sur l'appariement des entrées et des sorties des services, et ne traite pas les différentes facettes (non-fonctionnelle, technique, contextuelle, etc.) de vérification de composabilité entre deux ou plusieurs services à connecter.

Aussi, nos recherches ont-elles été motivées par l'absence, à notre meilleure connaissance, d'un modèle de composabilité qui traite plusieurs facettes de cette

problématique, tout en étant aligné avec les standards de spécifications des services Web, à savoir, WSDL 2.0, SAWSDL (*Semantic Annotations for WSDL*) et WS-Policy (*Web Services Policy*)1.5.

## 1.3. Fondements de notre approche

Notre approche est dirigée essentiellement par l'utilisation des standards W3C pour la description des services Web. Ces standards constituent des formalismes largement adoptés par la communauté des services Web et l'industrie informatique, qui a tendance à utiliser des outils et des techniques compatibles avec les standards pour le traitement ou l'exploitation des produits logiciels.

Afin de mieux cerner la problématique de composabilité de ces composants logiciels, nous avons examiné systématiquement la partie descriptive d'un service Web dans plusieurs modèles et standards. Assurément, l'efficacité du processus de composabilité reste fortement dépendante de la richesse de l'information recueillie concernant les services à composer, mais aussi de la couverture des différents aspects descriptifs de ces derniers.

Dans ce contexte, notre approche propose d'abord un modèle de description de services Web multi-aspects qui s'aligne avec les spécifications W3C, à savoir, WSDL 2.0, SAWSDL et WS-Policy 1.5. Ce modèle vise essentiellement à capturer le maximum d'informations sur les services concernés. Il décrit les propriétés fonctionnelles, non-fonctionnelles, orientés données, contextuelles et techniques d'un service Web à des niveaux syntaxique et sémantique, tout en assurant une totale conformité avec les spécifications desdits standards. Sur la base de ce modèle descriptif, notre approche considère un ensemble d'informations pertinentes pour la composabilité offline, objet de notre recherche. Ces informations permettent à la fois de vérifier la composabilité offline fonctionnelle, non-fonctionnelle, contextuelle, orientée données, et technique aux niveaux syntaxique et sémantique de deux ou plusieurs opérations de services Web. Nous avons élaboré un ensemble de règles de composabilité de services Web. Pour vérifier la possibilité d'interconnecter chaque couple d'opérations, ces règles comparent leurs propriétés descriptives dans le but de détecter d'éventuelles hétérogénéités. Elles forment le noyau de l'algorithme de composabilité offline que nous avons développé.

## 1.4. Contributions de la thèse

Les principales contributions de la thèse sont :

- ✓ Une analyse comparative fine des propriétés descriptives des services Web fournies d'un côté, par les standards W3C (WSDL 2.0, SAWSDL et WS-Policy 1.5) et d'un autre côté, par les deux modèles sémantiques OWL-S et WSMO.
- ✓ Un modèle de description de services Web multi-aspects aligné avec les standards W3C (WSDL 2.0, SAWSDL et WS-Policy 1.5) qui couvre les cinq classes de propriétés de services : fonctionnelles, non-fonctionnelles, contextuelles, orientées données et techniques.
- ✓ Une approche de composabilité offline de services Web, identifiant les informations descriptives impliquées dans l'évaluation de la connexion de deux opérations, et comprenant un ensemble de règles élaborées, un algorithme de composabilité développé et une expérimentation qui vise à mieux illustrer ladite approche.

## 1.5. Organisation de la thèse

Outre l'introduction exposée dans ce chapitre, ce rapport comprend cinq autres chapitres. Les deux premiers chapitres sont consacrés à l'exploration de l'existant en matière de représentation et composition de services Web, tandis que les chapitres 4, 5 et 6 détaillent notre contribution.

Le chapitre 2 présente une analyse comparative approfondie qui explore les mécanismes de description des services Web offerts, d'un côté, par les standards W3C (WSDL 2.0, SAWSDL et WS-Policy 1.5) et d'un autre côté, par les deux modèles sémantiques OWL-S et WSMO. Ces mécanismes sont présentés selon cinq classes de propriétés de services : fonctionnelles, non-fonctionnelles, contextuelles, orientées données et techniques.

Le chapitre 3 introduit le paradigme de composition de services Web en présentant les définitions et les types de composition, ainsi que les principales phases du cycle de vie d'un service composite. Il expose aussi, une revue des approches adoptées pour la

réalisation des deux phases clés de la composition des services Web, à savoir les phases de découverte et de génération de plans de composition.

Le chapitre 4 propose une approche « *Bottom-Up* » de description de services Web, élaborée suite à l'étude approfondie des grammaires des trois standards W3C (WSDL 2.0, SAWSDL et WS-Policy 1.5). Cette approche nous a permis de définir un modèle UML qui intègre les propriétés descriptives prévues par ces trois standards et les enrichit par de nouvelles propriétés, dans le but de capturer le maximum d'informations sur un service Web, tout en restant conforme aux standards. Les propriétés proposées couvrent les aspects fonctionnel, non-fonctionnel, orientés données, contextuel et technique. Elles permettent d'identifier chaque service Web d'une manière précise et non-ambiguë.

Le chapitre 5 se focalise sur le modèle de composabilité offline proposé pour vérifier la possibilité de connecter deux ou plusieurs opérations de services à travers un ensemble de règles conçues. Il introduit d'abord, la notion de composabilité des services Web et discute les limites des approches existantes. Ensuite, il présente les fondements de notre approche tout en identifiant les types d'informations pertinentes pour la composabilité offline fonctionnelle, non-fonctionnelle, contextuelle, orientée données, et technique aux niveaux syntaxique et sémantique. Puis, il détaille l'ensemble de règles de composabilité de services Web, élaborées pour permettre de comparer leurs propriétés descriptives et de détecter d'éventuelles hétérogénéités.

Enfin, le chapitre 6 est consacré à la présentation de la démarche globale et des mécanismes adoptés pour vérifier la composabilité offline de deux opérations. Il présente ensuite l'algorithme de composabilité offline développé comprenant les fonctions de vérification de composabilité relatives à chaque aspect . Ce chapitre termine par une expérimentation qui vise à mieux illustrer ladite démarche et tester l'algorithme conçu.

En conclusion, nous analysons les apports de notre approche et dégageons les principales perspectives considérées comme une suite logique de la présente recherche.

# Chapitre 2

# Etat de l'Art de la description des services Web

## Sommaire


***2.1. Introduction*** ***21***

***2.2. Propriétés des services Web*** ***22***

2.2.1. Propriétés fonctionnelles 23

2.2.2. Propriétés non-fonctionnelles 24

2.2.3. Propriétés contextuelles 24

2.2.4. Propriétés orientées données 24

2.2.5. Propriétés techniques 25

2.2.6. Propriétés comportementales 25

2.2.7. Synthèse 26

***2.3. Classification des approches de description des services Web*** ***26***

2.3.1. Niveaux de description syntaxique et sémantique 27

2.3.2. Approches à base de langages syntaxiques 28

2.3.3. Approches à base de modèles sémantiques 28

***2.4. Standards W3C : WSDL, SAWSDL et WS-Policy*** ***28***

2.4.1. Web Services Description Language (WSDL) 30

2.4.2. Semantic Annotations for WSDL (SAWSDL) 32

2.4.3. Web Services Policy (WS-Policy) 34

***2.5. Modèles sémantiques : OWL-S et WSMO*** ***36***

2.5.1. Ontology Web Language for Service (OWL-S) 37

2.5.2. Web Services Modelling Ontology (WSMO) 39

***2.6. Analyse des approches de description des services Web*** ***42***

2.6.1. Couverture de la description fonctionnelle 42

2.6.1.1. Cas des langages WSDL2.0 et SAWSDL ..... 42
2.6.1.2. Cas de l'ontologie OWL-S ..... 43
2.6.1.3. Cas de l'ontologie WSMO ..... 43
2.6.1.4. Discussion ..... 44
2.6.2. Couverture de la description de données ..... 46
2.6.2.1. Cas de WSDL/SAWSDL ..... 46
2.6.2.2. Cas de OWL-S et WSMO ..... 47
2.6.2.3. Discussion ..... 48
2.6.3. Couverture de la description non-fonctionnelle ..... 48
2.6.3.1. Cas de WSDL/WS-Policy ..... 49
2.6.3.2. Cas de OWL-S et WSMO ..... 50
2.6.3.3. Discussion ..... 50
2.6.4. Couverture de la description technique ..... 51
2.6.4.1. Cas de WSDL ..... 51
2.6.4.2. Cas de OWL-S et WSMO ..... 52
2.6.4.3. Discussion ..... 52
***2.7. Synthèse ..... 52***

## 2.1. Introduction

Le paradigme services Web est fondé sur la publication des services dans un annuaire qui rend accessibles les informations nécessaires à leur utilisation. Cet annuaire constitue un registre mettant à la disposition de l'utilisateur un ensemble d'informations qui décrivent les propriétés des services Web accessibles à travers le réseau Internet. Il indique principalement ce que fait un service, comment et où y accéder dans le but d'assurer un accès au service conditionné par un échange clair et structuré entre le client du service et son fournisseur. Toutefois, outre les propriétés fonctionnelles, un service Web peut posséder d'autres types de propriétés, telle que la qualité de service à titre d'exemple, qui font de ce composant logiciel le service le plus adapté à un client donné qu'à un autre. Une multitude de services Web, fournis par différents fournisseurs de services, proposent notamment des fonctionnalités similaires mais répondent différemment aux besoins particuliers des clients.

Aussi, une description complète voire sémantique des services Web forme-t-elle un élément clé pour favoriser une utilisation efficace et efficiente des services publiés en fonction des besoins spécifiques des clients qui peuvent être aussi bien de simples utilisateurs que des processus automatiques. Autrement dit, disposer au préalable d'un ensemble d'informations précises sur les différentes propriétés d'un service Web, telles que, la ou les fonctionnalités offertes, ses contraintes techniques, la structure des données de ses entrées et sorties, son processus interne, est indispensable pour assister aussi bien un client humain qu'un processus automatique de découverte, de sélection ou de composition des services Web. Il devrait permettre de choisir des services appropriés, vérifier leur composabilité (i.e. la possibilité de les composer) et de s'informer sur la manière de les composer quand il s'agit de répondre à une requête complexe.

Dans ce contexte, nous considérons que l'efficacité du processus de composition en particulier dépend étroitement du degré de précision des descriptions des services Web. Nous pensons que des compositions efficientes de services Web, fondées d'abord sur la vérification de leur composabilité, ne peuvent réellement être réalisées que si les hétérogénéités des descriptions des services ont été détectées et résolues. Aussi, ce chapitre est-il consacré à l'état de l'art de la description des services Web. Nous y

définissons d'abord les propriétés de ces services sous forme de classes relatives à leurs différents aspects fonctionnel, non fonctionnel, contextuel, orienté données, technique et comportemental. Nous classifions et présentons ensuite les approches proposées dans la littérature pour leur description. Celles-ci décrivent les services Web à deux niveaux syntaxique et/ou sémantique et peuvent être principalement réparties en standards W3C, WSDL 2.0 [Chinnici et al., 2007], SAWSDL [Farrell et al., 2007], WS-Policy [Vedamuthu et al., 2007a] et modèles sémantiques OWL-S [Martin et al., 2004a] et WSMO [Roman et al., 2006]. Nous terminons par une analyse comparative qui a fait l'objet d'un article de synthèse [Omrana et al., 2013b] ayant permis d'évaluer le degré de couverture des différents aspects des services Web par les approches de description dédiées. La synthèse qui conclut ce chapitre souligne l'intérêt particulier des standards W3C et justifie leur choix par la suite comme fondement de notre approche de vérification de la composabilité des services Web.

## 2.2. Propriétés des services Web

Les propriétés que possède un service Web sont diverses et assez diversifiées de par les aspects qu'elles couvrent. Elles peuvent aller d'une simple information sur ce que fait un service Web jusqu'au détail sur la manière de le faire et avec quelle garantie du point de vue sécurité ou qualité de service, sans oublier les détails techniques nécessaires pour l'accès concret au service.

Dans ce contexte, un tour d'horizon et une synthèse des approches de descriptions des services proposées dans la littérature nous ont permis de dégager six classes de propriétés des services Web (Figure 2.1). Ces classes se distinguent l'une de l'autre par l'aspect qu'elles décrivent du service Web. Elles se répartissent en propriétés fonctionnelles, non-fonctionnelles, contextuelles, orientées données, techniques et enfin comportementales. Chacune des classes précitées enrichit la description du service Web par un type d'information bien particulier que nous détaillons dans la suite de ce paragraphe. L'information fournie a une forme syntaxique mais aussi un sens. Aussi, peut-elle être décrite à deux niveaux : syntaxique et sémantique.

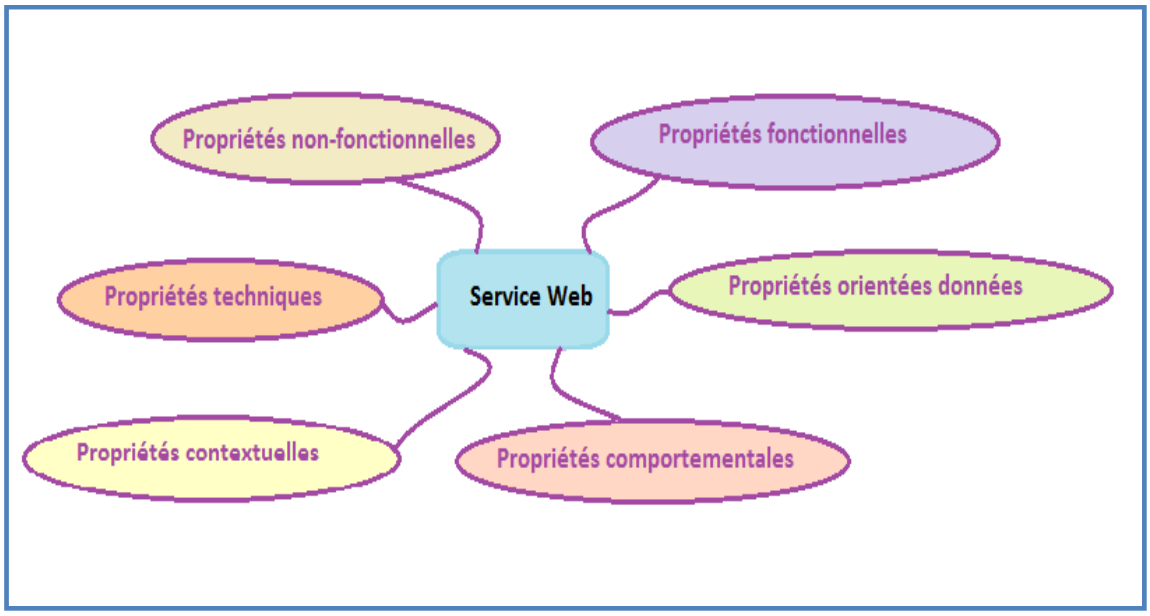


**Figure 2.1**. Les six classes de propriétés descriptives du service Web

### 2.2.1. Propriétés fonctionnelles

La classe de propriétés fonctionnelles comprend un ensemble de propriétés qui constituent une spécification de ce que peut offrir un service Web à ses clients en termes de fonctionnalités, lorsqu'il est invoqué. Elle renseigne sur son aspect fonctionnel, désigné dans la littérature, par la capacité de ce service. Aussi, la classe des propriétés fonctionnelles des services Web comprend-t-elle des éléments qui décrivent leurs capacités en considérant toutes les conditions préalables à leur utilisation aussi bien que les résultats qu'ils fournissent.

En résumé, une propriété fonctionnelle est une propriété relative à la fonctionnalité du service Web qu'elle décrit [Vitvar et al., 2009]. Elle donne une information précise sur son usage. Elle peut indiquer à titre d'exemple une entrée, une sortie, une opération ou une catégorie métier du service Web décrit, ou encore une pré-condition à vérifier au préalable avant son exécution. Dans le cas d'un service Web de paiement électronique par exemple, « payer en ligne par carte bancaire », « vérifier la validité du numéro de la carte bancaire » et « paiement électronique » sont trois propriétés fonctionnelles qui renseignent respectivement sur l'opération qu'il exécute, la pré-condition à vérifier au préalable avant son exécution et sa catégorie métier.

### 2.2.2. Propriétés non-fonctionnelles

La classe de propriétés non-fonctionnelles comprend un ensemble de propriétés qui décrivent l'aspect non-fonctionnel du service Web. Ces propriétés sont souvent désignées comme des contraintes, des attributs ou tout simplement des paramètres du service Web qui n'informent pas sur son aspect fonctionnel. Elles constituent en fait des paramètres qui régissent son comportement ou son utilisation tels que la qualité de service, la fiabilité, la performance, la sécurité, ou encore le prix [(Chung et al., 1999), (Singhera, 2004)]. Dans le cas d'un service Web de paiement électronique à titre d'exemple, la description des propriétés non-fonctionnelles reliées à la sécurité est primordiale. La signature numérique, le type de cryptage et le niveau de confiance sont par exemple trois propriétés non-fonctionnelles qui peuvent être indiquées pour renseigner sur le niveau de sécurité garanti par le service Web fourni.

### 2.2.3. Propriétés contextuelles

Un autre aspect important des services Web est relatif au contexte de leurs propriétés fonctionnelles et non-fonctionnelles. Il relève de leur contexte d'utilisation et comprend des propriétés telles que l'unité et l'échelle qui doivent être associées à des valeurs de propriétés fonctionnelles ou non-fonctionnelles. Si nous considérons à titre d'exemple le cas d'un service Web qui exécute une opération ayant comme entrée le prix d'un article. Des attributs d'ordre contextuel, tels que la devise et le facteur multiplicateur associés à l'entrée « *Prix* », demeurent nécessaires pour interpréter correctement les données échangées avec le service Web en question.

### 2.2.4. Propriétés orientées données

La classe des propriétés orientées données comprend les types des données utilisées par le service Web, ainsi que toute information relative à sa mise en correspondance avec un autre type de données (fonction, schéma de « mapping », etc.). Ces informations demeurent primordiales pour une utilisation correcte du service mais aussi pour sa composition avec d'autres services. Elles permettent d'une part, de pouvoir communiquer correctement avec le service en utilisant des données compatibles avec les structures qu'il utilise, et d'autre part, d'être averti quand il est question de composer le service avec un autre qui n'utilise pas les mêmes types de données et qui nécessite par

la suite de réaliser une mise en correspondance entre des types de données hétérogènes. Nous pouvons citer ici le cas d'un service Web qui fournit comme sortie le prénom et nom d'un auteur sous forme d'une chaîne unique de caractères et qui doit être composé avec un autre service Web ayant pour entrée la même information fournie sous forme d'une structure composée de deux champs prénom et nom.

Les propriétés orientées données peuvent être reliées à tout élément utilisé dans la description des aspects d'un service Web pour décrire sa structure de données (qui peut être indifféremment un type simple ou une structure complexe) et son « *mapping* » de ou vers d'autres structures de données. Aussi, peuvent-elles à titre d'exemples être appliquées aux entrées et sorties des opérations de services comme aux attributs de qualité de service tels que le temps d'exécution et l'unité de mesure considérée (seconde, milliseconde ou autre).

### 2.2.5. Propriétés techniques

Les propriétés techniques sont des propriétés qui informent sur l'aspect technique du service Web. Il est à noter que ces propriétés doivent nécessairement être connues au préalable par le client avant de pouvoir utiliser le service qu'il souhaite invoquer. Elles lui permettent d'une part, de savoir accéder au service, et d'autre part, de pouvoir communiquer correctement avec ce service en échangeant des messages dont la structure et les protocoles sont conformes à ce qui est proposé par ce même service.

En résumé, outre le port ou point d'accès au service, une propriété technique fournit également un détail technique sur l'échange de messages avec le service Web qu'elle décrit. Comme exemples de propriétés de ce type, nous citons le protocole de communication utilisé, le point d'accès au service (*Endpoint*) et les détails techniques des messages échangés tels que le format et la sérialisation des messages.

### 2.2.6. Propriétés comportementales

Les propriétés comportementales sont des propriétés qui renseignent sur le comportement du service Web. Ce type de propriétés est particulièrement essentiel dans le cadre de la composition des services Web. Ces informations sur le comportement interne et externe du service composite permettent d'une part, d'exécuter dans l'ordre

prévu ses services internes quand ce service Web est invoqué, et d'autre part, d'interagir correctement avec des services externes à ce service composite.

Les propriétés comportementales d'un service Web fournissent des détails sur un aspect important de ce service et doivent nécessairement être mentionnées dans sa description. Comme cet aspect est étroitement lié à la problématique que nous abordons dans cette recherche, nous lui consacrons un paragraphe dans le chapitre suivant réservé à l'état de l'art de la composition des services Web.

### 2.2.7. Synthèse

Les services Web possèdent des propriétés diverses que nous répartissons en six classes : propriétés fonctionnelles, non-fonctionnelles, contextuelles, orientées données, techniques et enfin comportementales. Chacune de ces classes apporte une information qui peut être fournie sous sa forme syntaxique, voire être décrite par des détails sémantiques qui renseignent sur son sens. Nous considérons que toutes ces classes sont fondamentales pour assurer une description exhaustive des services Web et que leur description sémantique est primordiale pour permettre en plus d'automatiser leur traitement. Dans le cadre de ce travail en particulier, décrire l'ensemble de ces propriétés, aux niveaux syntaxique et sémantique, devrait nous permettre d'évaluer avec précision les services Web et par la suite de vérifier automatiquement et correctement leur composabilité.

## 2.3. Classification des approches de description des services Web

Les propriétés des services Web peuvent être décrites à deux niveaux : syntaxique et/ou sémantique. Les premières spécifications de services Web [Chinnici et al., 2002] ont mis l'accent sur leur aspect fonctionnel décrit à un niveau syntaxique seulement. Toutefois, le besoin imminent d'automatisation des processus de découverte, de sélection et de composition de services suite à leur prolifération exponentielle sur le Web, a très rapidement révélé les limitations de ces travaux.

En effet, la plupart des services publiés sur Internet se trouvent décrits par des fichiers de type WSDL. Toutefois, ce langage qui constitue un standard W3C s'est avéré inapte à fournir une description suffisamment riche pour les consommateurs des services Web.

Il se contente de décrire au niveau syntaxique des aspects élémentaires des services Web. Ainsi, les standards W3C, SAWSDL et WS-Policy, ont-ils été proposés dans la littérature pour pallier cette limitation. Ces approches introduisent d'autres propriétés de services Web et prennent également en considération le niveau de description sémantique en enrichissant les fichiers WSDL.

D'autres travaux et modèles sémantiques [(Fensel et al., 2002), (Martin et al., 2004a), (Akkiraju et al., 2005), (Battle et al., 2005), (Pathak et al., 2006), (Roman et al., 2006 ) , (Badr et al., 2008)] ont essayé de pallier ces limites en introduisant l'aspect non-fonctionnel et en proposant des descriptions de niveau sémantique.

Une analyse fondée sur les caractéristiques et fondements 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 [Martin et al., 2004a] et WSMO [Fensel et al., 2002], nous a permis d'en distinguer deux classes. La première classe est fondée sur l'utilisation de langages syntaxiques à base de XML pour décrire les services Web et la deuxième classe utilise par contre des modèles sémantiques pour ce faire. Les deux classes sont présentées dans cette section à la suite du paragraphe exposant les niveaux de description des services Web.

### 2.3.1. Niveaux de description syntaxique et sémantique

Le niveau de description syntaxique des propriétés de services Web informe sur la syntaxe des mots utilisés pour désigner ces propriétés. A ce niveau, les propriétés sont analysées et appariées en se basant sur des techniques d'analyse syntaxique. Par exemple, les deux opérations de services « Achat de billets d'avion » et « Acheter des billets d'avion » sont syntaxiquement équivalentes : le mot «Achat» serait reconnu comme forme fléchie (nom dérivé) du verbe «Acheter».

Par contre, le niveau de description sémantique des propriétés de services Web renseigne sur le sens de ces propriétés et non sur leur syntaxe, dans le but de procurer une description plus précise et interprétable par la machine. Examinons le cas de deux entrées d'opérations de services Web « Auteur » et « Ecrivain ». Ces deux propriétés fonctionnelles sont syntaxiquement différentes mais sémantiquement équivalentes. Le niveau de description sémantique est fortement sollicité dans les appariements de

services Web de fournisseurs différents. L'information sémantique peut être définie au niveau d'un modèle sémantique comme l'ontologie.

### 2.3.2. Approches à base de langages syntaxiques

La classe des approches à base de langages syntaxiques utilise des structures syntaxiques de type XML pour décrire les propriétés des services Web. Elle est constituée du langage WSDL mais aussi des extensions de ce langage à savoir SAWSDL et les WS*-spécifications tels que WS-Policy. L'ensemble de ces langages possède le qualificatif standard. Ils ont été proposés par la communauté de chercheurs du consortium W3C afin de munir le langage ancêtre WSDL d'une couverture plus large des différents aspects de description de services Web tout en prenant compte de leur sémantique au moyen du langage SAWSDL. Outre le fait qu'ils soient standardisés, leur structure basée sur le langage XML en fait des langages facilement extensibles pour s'adapter à de nouveaux et éventuels besoins en matière de description des services Web. Nous présentons en détail chacun de ces standards dans la section 2.4.

### 2.3.3. Approches à base de modèles sémantiques

La classe des approches fondées sur l'utilisation des modèles sémantiques adopte une démarche sémantique pour décrire les services Web. Dans le souci de fournir une description sémantique de ces services interprétable par la machine, elle opte pour l'exploitation de la notion d'ontologie qui a été adoptée par plusieurs travaux de recherche en Web sémantique en tant qu'approche prometteuse pour ce type de traitement. Cette classe comprend les deux frameworks OWL-S et WSMO qui représentent les modèles sémantiques de description des services Web les plus référencés. Ces frameworks proposent une description enrichie des services couvrant plusieurs aspects, mais ils ne se limitent pas uniquement à la description de ces services. Ils proposent aussi des mécanismes de découverte et de composition automatiques des services Web.

## 2.4. Standards W3C : WSDL, SAWSDL et WS-Policy

La description syntaxique de services Web a atteint un niveau de maturité qui s'est traduit par l'émergence d'une collection de standards de *World Wide Web Consortium*

(W3C). Le W3C s'est imposé, ces dernières années, comme organisme producteur de standards de services Web qui sont largement reconnus par la communauté industrielle.

A présent, la couche syntaxique des développements existants relatifs à la technologie SOA, est assurée principalement par WSDL, le standard de W3C. Ce standard offre une description fonctionnelle et technique de services Web qui informe essentiellement sur les structures de données échangées. Cependant, cette description ne couvre pas les autres aspects non-fonctionnel et comportemental. Elle se limite, en fait, au niveau syntaxique qui décrit la structure des messages à échanger avec le service. Le standard W3C, SAWSDL, considéré dans la littérature comme une approche hybride entre le syntaxique et le sémantique [Baltá et al., 2012], est une extension du standard WSDL qui propose des mécanismes d'annotation sémantique des éléments (propriétés) décrits par ce dernier. Il permet d'étendre la description syntaxique offerte par WSDL à un niveau sémantique.

Par ailleurs, W3C propose également un autre standard, nommé WS-Policy, qui permet de décrire aussi un service Web par les politiques adoptées par son fournisseur. Ces politiques décrivent particulièrement des propriétés non-fonctionnelles des services Web relatives à un domaine non-fonctionnel donné tels que la sécurité et la qualité de service. Outre cet aspect des services Web, W3C préconise également l'utilisation de la recommandation WS-CDL (*Web Services Choreography Description Language*) [Kavantzas et al., 2004] pour couvrir leur description comportementale. Cette spécification décrit les collaborations point à point des différents participants, d'un point de vue global (le comportement externe du service). Concernant le comportement interne du service (lié à l'orchestration des services), OASIS (*Organization for the Advancement of Structured Information Standards*) propose le standard WS-BPEL (*Web Services Business Process Execution Language*) [Alves et al., 2007].

La suite de cette section introduit les standards WSDL 2.0, SAWSDL et WS-Policy 1.5 dans le but d'expliciter des notions relatives à la description des services Web, utilisées plus loin dans ce rapport.

### 2.4.1. Web Services Description Language (WSDL)

Le standard W3C WSDL est considéré comme le langage « *de facto* » de description des services Web. Il définit un service Web en décrivant d'une part, son interface abstraite qui agrège un ensemble d'opérations assurant une fonctionnalité donnée, et d'autre part, une ou plusieurs liaisons (*Binding*) concrètes qui informent sur les détails techniques de son utilisation à travers cette même interface. Sa nouvelle version, WSDL 2.0 [Chinnici et al., 2007], assure la réutilisation, l'héritage et l'homogénéité entre ses différents composants.

```
<?xml version="1.0" encoding="UTF-8"?>
<description xmlns="http://www.w3.org/ns/wsdl"
             xmlns:tns="http://www.tmsws.com/AchatLivre"
             xmlns:whttp="http://schemas.xmlsoap.org/wsdl/http/"
             xmlns:wsoap="http://schemas.xmlsoap.org/wsdl/soap/"
             targetNamespace="http://www.tmsws.com/AchatLivre">

<!-- Définir les différents types -->
   <types>
      <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                 xmlns="http://www.tmsws.com/wsdl20sample"
                 targetNamespace="http://www.example.com/A">
   ....
    </types>
<!-- Définir l'interface du service Web -->
   <interface name="AchatLivre">
      <fault name="ClientError" element="tns:string"/>
      <fault name="ServerError" element="tns:string"/>
      <fault name="Redirection" element="tns:string"/>

      <operation name="RechercherParISSN" pattern="http://www.w3.org/ns/wsdl/in-out">
         <input messageLabel="ISSN" element="tns:string"/>
         <output messageLabel="Prix" element="tns:string"/>
      </operation>
   .....
 <!-- Définir les Bindings  -->
   <binding name="InterfaceHttpBinding" interface="tns:AchatLivre"
            type="http://www.w3.org/ns/wsdl/http">
      <operation ref="tns:RechercherParISSN" whttp:method="GET"/>
   ....
   </binding>

 <!-- Définir les accès au service Web -->
   <service name="RESTfulService" interface="tns:tns:AchatLivre">
      <endpoint name="RESTfulServiceHttpEndpoint"
                binding="tns:RESTfulInterfaceHttpBinding"
                address="http://www.AchatLivre.com"/>
    .....
         </service>
</description>
```

**Figure 2.2**. Extrait du fichier WSDL 2.0 décrivant le service « ***AchatLivre*** »

Pour illustrer l'ensemble des éléments WSDL évoqués ci-dessus, nous prenons l'exemple du service Web d'achat électronique de livres « *AchatLivre* ». Ce service est décrit par l'interface *AchatLivre* qui définit trois types de messages d'erreurs *ClientError*, *ServerError* et *Redirection,* et l'opération *RechercherParISSN* qui a comme entrée *ISSN* et comme sortie *Prix*. Cette interface est concrétisée par le biais du Binding *InterfaceHttpBinding* et le service est accessible à travers l'endpoint *http://www.AchatLivre.com*. La figure 2.2 montre un extrait de la description de ce service en illustrant ces éléments de description.

Nous détaillons, dans ce qui suit, les quatre éléments fondamentaux de ce langage, à savoir, l'interface, le type, la liaison et le service, conformément à cette spécification.

- **Interface :** Cet élément décrit la ou les fonctionnalités offertes par le service Web à travers les opérations qu'il exécute. Il est défini comme un ensemble d'opérations et de messages d'erreurs dont chacun peut correspondre à une ou un ensemble de ces opérations. Chaque opération est elle même définie par un ensemble d'entrées, de sorties et de messages d'erreurs en entrée ou en sortie. La description fournie par l'interface demeure une description abstraite du service Web qui définit ses fonctionnalités sans donner les détails techniques nécessaires à savoir pour pouvoir y accéder. Ces détails sont fournis par les éléments liaison et service qui seront décrits par la suite.
- **Type :** Cette propriété correspond au type de données d'une entrée, d'une sortie ou d'un message que le service peut envoyer ou recevoir. Il définit le schéma de données utilisé pour représenter cet élément. Il est souvent spécifié en utilisant un schéma de données XML (XSD) [Arkin et al., 2004]. Sa spécification peut être explicitement définie à l'intérieur du document WSDL ou importée à partir d'un document de schéma de données externe.
- **Liaison (*Binding*) :** Cet élément spécifie la manière et les règles qui régissent l'accès aux fonctionnalités (opérations) décrites par l'interface du service Web. Il fournit notamment des informations tels que le type du protocole de transport des messages (par exemple, HTTP, SOAP ou MIME), leur encodage (par exemple, texte ou binaire) et leur sérialisation. Les propriétés du binding d'un service Web peuvent différer selon le point d'accès (port, appelé *Endpoint* dans la spécification WSDL 2.0) utilisé pour accéder à ce service. Aussi, chaque

*Binding* est associé à une interface unique définie par l'interface abstraite du service mais aussi son point d'accès.

- **Service:** Cette propriété définit les différents points d'accès (*Endpoints*) au service. Du point de vue fonctionnel, un service répond à une requête spécifique donnée. Cependant, ce même service peut être accessible différemment (suivant différents *Bindings*) à travers différents *Endpoints*. Par conséquent, un service WSDL 2.0 est perçu comme une agrégation d'*Endpoints* qui permettent d'accéder à ses fonctionnalités avec différentes liaisons. Chaque *Endpoint* est associé à un *Binding* et indique une adresse URI d'accès au service Web. Finalement, chaque service implémente une et une seule interface mais avec peut être différents points d'accès et par la suite différentes propriétés non-fonctionnelles.

Finalement, la spécification WSDL 2.0 joue un rôle important dans l'interopérabilité des services Web. Elle est définie selon une sémantique totalement indépendante du modèle de programmation de l'application. En effet, elle sépare clairement la définition abstraite du service (en termes de messages à échanger avec le service), de ses mécanismes de liaison tels que la définition des protocoles applicatifs.

### 2.4.2. Semantic Annotations for WSDL (SAWSDL)

SAWSDL [Farrell et al., 2007] est un standard W3C qui constitue une extension de WSDL, dont le but est d'augmenter l'expressivité de ce dernier en annotant sémantiquement certains de ses éléments (ceux qui possèdent une sémantique). Cette annotation est réalisée par le biais de deux éléments du SAWSDL, *ModelReference* et *SchemaMapping* ,et elle est intégrée (incorporée) dans le ficher WSDL.

L'attribut *ModelReference* permet d'annoter sémantiquement les types de données, les interfaces, les opérations, les messages d'erreurs, les entrées et les sorties décrits dans WSDL. L'annotation consiste à associer un ou plusieurs concepts sémantiques à chacun de ces éléments, tout en restant indépendant d'un langage ou d'un framework de représentation sémantique, autrement dit, d'un modèle sémantique [Akkiraju et al., 2007]. Notamment, l'annotation est réalisée en fournissant l'URI du concept sémantique qui peut être défini dans un modèle sémantique quelconque (ontologie ou

autre). Les aspects relatifs aux propriétés non-fonctionnelles (par exemple, la qualité) et le comportement (l'orchestration des services) ne sont pas traités par SAWSDL.

Par ailleurs, l'élément *SchemaMapping* est utilisé pour indiquer la correspondance (l'appariement) entre la structure de données XML d'un élément et celle du concept qui l'annote telle qu'elle est donnée dans le modèle sémantique où il est défini. Il possède deux attributs qui permettent d'attacher les schémas d'appariement *LiftingSchemaMapping* et *LoweringSchemaMapping* à la structure de données décrite. L'attribut *LiftingSchemaMapping* indique comment transformer les données XML en données conformes à un modèle sémantique, alors que l'attribut *LoweringSchemaMapping* montre comment transformer les données dans l'autre sens, d'un modèle sémantique à une structure XML.

```
<!-- Définir les différents types -->
   <types>
      <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                 xmlns="http://www.tmsws.com/wsdl20sample"
                 targetNamespace="http://www.example.com/A">

         <xs:element name="Prix"  type="tns:string"
         sawsdl:modelReference = "http://emi/AchatOntology/#Prix"
         sawsdl:loweringSchemaMapping = "http://emi/AchatOntology/Ont2PriceType.xml"/
         sawsdl:liftingSchemaMapping = "http://emi/AchatOntology/PriceType2Ont.xml">
    </types>

<!-- Définir l'interface du service Web -->

   <interface name="AchatLivre" sawsdl:modelReference="http://emi/AchatOntology/#AchatLivre">

      <fault name="ClientError" element="tns:string"/>
      <fault name="ServerError" element="tns:string"/>
      <fault name="Redirection" element="tns:string"/>

      <operation name="RechercherParISSN" pattern="http://www.w3.org/ns/wsdl/in-out"
      sawsdl:modelReference ="http://emi/AchatOntology/#RechercherParISSN">
         <input messageLabel="ISSN" element="tns:string"/>
         <output messageLabel="Prix" element="tns:string"/>
      </operation>
```

**Figure 2.3**. Extrait du fichier SAWSDL décrivant le service « ***AchatLivre*** »

La figure 2.3 montre un extrait de la description sémantique du service « *AchatLivre* » conformément au langage SAWSDL. L'interface *AchatLivre*, l'opération *RechercherParISSN* et le type *Prix* sont annotées respectivement par les trois concepts ontologiques d'URI http://emi/AchatOntology/#AchatLivre,

http://emi/AchatOntology/#RechercherParISSN et http://emi/AchatOntology/#Prix . Les schémas de mapping *LoweringSchemaMapping* et *LiftingSchemaMapping* associés au type Prix sont définis respectivement par les fonctions d'URI http://emi/AchatOntology/Ont2PriceType.xml et http://emi/AchatOntology/PriceType2Ont.xml. Les éléments d'annotation utilisés sont illustrés par la figure 2.3.

SAWSDL se présente comme une approche hybride qui associe une description syntaxique à des modèles sémantiques. Son avantage majeur réside dans son alignement avec le standard WSDL. S'ajoute à cela le fait que les annotations sémantiques qu'il fournit demeurent indépendantes du ou des modèles sémantiques utilisés. Par ailleurs, la description sémantique des services Web existants (majoritairement, si ce n'est totalement, décrits au moyen de WSDL) reste relativement moins coûteuse avec SAWSDL qu'avec d'autres langages ou frameworks de description sémantique. Notamment, les outils de développement utilisés dans l'industrie pour traiter ou décrire des services Web en WSDL peuvent facilement s'adapter à son extension SAWSDL.

### 2.4.3. Web Services Policy (WS-Policy)

En plus de ses capacités fonctionnelles, un service Web possède des capacités non fonctionnelles comme son niveau de sécurité et sa qualité de service. Un fournisseur de services peut fournir le même service Web fonctionnel selon différentes politiques en termes d'aspects non fonctionnels. Un service Web donné peut par exemple être fourni avec différents niveaux de sécurité en fonction des protocoles de transport et des algorithmes de cryptage utilisés. Dans ce cas, le choix de ces protocoles et algorithmes forme une politique du service Web en termes de sécurité.

WS-Policy [Vedamuthu et al., 2007a], dont la dernière version à ce jour est WS-Policy1.5, est un standard W3C qui sert à décrire les politiques adoptées par le fournisseur d'un service Web aussi bien que les attentes d'un client (en termes de politiques souhaitées). Il utilise une grammaire syntaxique simple et extensible pour décrire et communiquer les stratégies d'un service Web. WS-Policy fournit notamment des constructions fondamentales qui peuvent être étendues par d'autres spécifications

des services Web. Ce standard est fondé sur XML dans le but de permettre de véhiculer les stratégies des services Web dans un mode interopérable.

Selon WS-Policy, la politique d'un service Web peut être exprimée par plusieurs politiques alternatives. Chaque politique alternative est constituée d'un ensemble d'assertions. Chaque assertion exprime une caractéristique (décrite par le fournisseur) ou une exigence (exigée par le client) d'un sujet qui peut être un *Endpoint*, un message, une opération, etc. Par exemple, l'assertion « *Encrypted*» associée à un endpoint indique que les messages échangés doivent être cryptés. Des opérateurs sont également utilisés pour exprimer les politiques. Nous en citons par exemple *ExactlyOne* ou *All*. L'opérateur *All* spécifie le fait que toutes les assertions constituant une politique alternative sont ou doivent être satisfaites pour accepter la politique alternative proposée. L'opérateur *ExactlyOne* spécifie qu'au moins l'une des politiques alternatives est ou doit être satisfaite pour accepter cette politique.

Par ailleurs, l'utilisation de l'élément WS-PolicyAttachment [Vedamuthu et al., 2007b] permet d'attacher une politique à un sujet comme partie intrinsèque de sa définition (dans le fichier WSDL ou au niveau de l'UDDI par exemple) ou encore comme partie d'un document indépendant (à part) utilisé à cette fin.

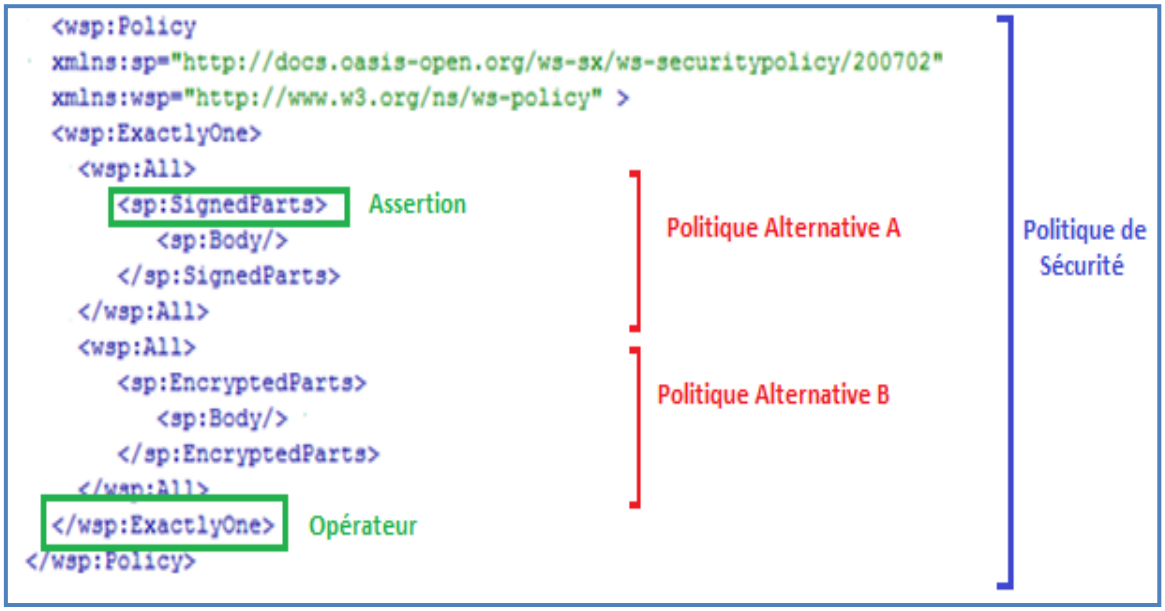


**Figure 2.4**. Politique de sécurité du service «***AchatLivre***» exprimée en WS-SecurityPolicy

La figure 2.4 montre un extrait de la description des propriétés non-fonctionnelles relatives à la sécurité du service « *AchatLivre* » au moyen du langage WS-Policy. Les assertions *EncryptedParts* et *SignedParts* permettent de spécifier les parties du message (échangé par le service Web) qui sont respectivement concernées par une protection de confidentialité (chiffrement) et d'intégrité (signature numérique). Dans l'exemple de la figure 2.4, c'est le corps (*Body*) du message qui est concerné par cette protection.

Les assertions utilisées pour décrire les politiques d'un service Web peuvent porter sur différents domaines non-fonctionnels tels que la sécurité ou la qualité de service. Certains domaines sont actuellement couverts par des spécifications définissant l'ensemble des assertions qui y sont relatives. Notamment, ces spécifications permettent de décrire de manière standard les propriétés des services Web relatives à un domaine non-fonctionnel précis. Nous citons à titre d'exemple la spécification WS-PolicySecurity [Nadalin et al., 2009] couvrant le domaine de sécurité. Par ailleurs, nous notons l'absence d'une description sémantique des assertions au niveau de WS-Policy. Cette limitation ne permet pas d'interpréter correctement des assertions non-fonctionnelles relatives à des domaines qui correspondent à des contextes utilisateurs spécifiques et qui ne se trouvent pas couverts par des spécifications standards tels WS-PolicySecurity.

## 2.5. Modèles sémantiques : OWL-S et WSMO

Le Web sémantique [Berners-Lee et al., 2001] vise à fournir, pour chaque ressource du Web, une sémantique interprétable par la machine. Dans ce contexte, les ontologies se présentent comme l'élément clé du Web sémantique. Elles ont pour but de décrire des concepts et leur interrelations, ainsi que les règles de déduction qui les régissent. De ce fait, elles sont considérées comme un moyen qui permet à l'utilisateur d'effectuer des recherches plus pertinentes sur le Web, vu qu'il devient capable d'accéder non seulement aux ressources liées syntaxiquement à sa requête, mais aussi à celles qui lui sont liées sémantiquement et qui peuvent être déduites à partir des ontologies de domaines.

Etant des ressources disponibles à travers le Web, les services Web devraient aussi être décrits sémantiquement afin de pouvoir automatiser leur découverte, sélection,

composition, etc. Les services Web sémantiques sont notamment le résultat de la combinaison des technologies des services Web et du Web sémantique, en particulier les ontologies. Dans ce cadre, les approches OWL-S et WSMO représentent les deux spécifications de description de services Web sémantiques les plus référencées et les plus utilisées dans la littérature. Elles proposent une description sémantique des services Web et des mécanismes d'invocation, de découverte et de composition automatiques de ces services. Dans ce qui suit, nous introduisons les principaux composants de OWL-S et WSMO.

### 2.5.1. Ontology Web Language for Service (OWL-S)

Appelé DAML-S dans ses premières versions [Ankolekar et al. , 2003], le langage OWL-S (OntologyWeb Language for Service) [(Martin et al., 2004a), (Norton, 2005)], dont la dernière version à ce jour est OWL-S 1.2, a pour objectif de décrire sémantiquement les propriétés d'un service Web de façon claire et interprétable par la machine. Outre la description des propriétés de base, il fournit une spécification des pré-requis et des conséquences d'exécution du service ainsi qu'un mécanisme de description des services composites et des flux de données. Ce langage est fondé sur l'utilisation du langage d'ontologie du Web (OWL) [Martin et al., 2004b] connu comme langage de référence dans le domaine des langages sémantiques utilisant la logique de description.

OWL-S est considéré comme une ontologie OWL particulière qui fournit des dispositifs spécifiques pour la description, la publication, la découverte, la composition et l'invocation des services Web. Ces dispositifs visent à automatiser le plus possible, la gestion des services Web. Dans ce contexte, OWL-S structure notamment la description d'un service Web en trois sous-ontologies: le profil du service (*ServiceProfile*),son modèle processus (*ServiceModel*) et ses liaisons (*ServiceGrounding*).

- **ServiceProfile :** Cette ontologie montre ce que le service fait. Elle décrit de façon précise les propriétés aussi bien fonctionnelles que non-fonctionnelles des services Web, de sorte que les clients puissent facilement identifier ceux qui répondent le mieux à leur besoin. Notamment, OWL-S donne aux fournisseurs de services la possibilité de créer leurs propres sous-classes spécialisées de la classe *Profile* afin de spécifier avec précision leurs services.

Par ailleurs, l'ontologie *ServiceProfile* pourrait être utilisée aussi bien par les fournisseurs pour associer des descriptions aux services publiés, que par les utilisateurs de services pour désigner leurs critères de recherche [Norton, 2005]. La figure 2.5 montre un extrait du profil de l'exemple des service Web d'achat électronique de livres « *AchatLivre* », décrit au moyen du langage OWL-S. l'ontologie *ServiceProfile* prévoit des propriétés non-fonctionnelles telles que « *contactInformation* » et « *qualityRating* » pour compléter l'information fonctionnelle définie par « *hasInput* » et « *hasOutput* », mais aussi par « *hasPrecondition* » et « *hasEffect* » (cf. Figure 2.5).

```
<profile:serviceName> AchatLivre </profile:serviceName>
    <profile:textDescription>
        Service Web d'achat électronique de livres
    </profile:textDescription>

    <profile:contactInformation>
      <actor:Actor rdf:ID="Achat_contacts">
        <actor:name>Responsable Achat</actor:name>
        <actor:phone>(212) 05678888 </actor:phone>
                <actor:email>achatlivre@emi.ma</actor:email>
        </actor:Actor>
    </profile:contactInformation>

    <!-- Quality Rating -->
    <profile:qualityRating>
      <profile:QualityRating rdf:ID="AchatLivre-Rating">
    <profile:ratingName>
      GoodRating
    </profile:ratingName>
    <profile:rating rdf:resource="&concepts;#GoodRating"/>
      </profile:QualityRating>
    </profile:qualityRating>

    <profile:hasInput rdf:resource="&AchatLivreProcess;#ISSN"/>
    <profile:hasPrecondition rdf:resource="&AchatLivreProcess;#LivreExiste"/>
    <profile:hasEffect rdf:resource="&AchatLivreProcess;#LivraisonLivreEffect"/>
    <profile:hasOutput rdf:resource="&AchatLivreProcess;#Prix"/>
```

**Figure 2.5.** Extrait du profil du service Web « ***AchatLivre*** » décrit par OWL-S

- **ServiceModel :** L'ontologie *ServiceModel* répond à la question « comment utiliser le service ». Elle décrit la manière avec laquelle un client doit interagir avec le service qu'il souhaite utiliser. Dans ce contexte, OWL-S modélise un

service Web en tant que processus qui peut être de type atomique (*AtomicProcess*), simple (*SimpleProcess*) ou composite (*CompositeProcess*). Un processus atomique est la description d'une action (opération) que le service peut effectuer en une seule interaction avec le service client. Il représente l'unité fondamentale la plus fine pour la mise en œuvre d'un processus global. Un processus simple fournit, à travers un mécanisme d'abstraction, des vues spécialisées de processus atomiques ou des vues simplifiés de processus composites à des fins de planification ou de raisonnement. Le processus composite est décomposable en d'autres processus (atomiques ou composites) qui sont composés en utilisant des constructions de contrôle telles que « Si-Alors-Sinon » ou des constructions de « séquence » et de « jointure ».

- **ServiceGrounding :** L'ontologie *ServiceGrounding*, quant à elle, indique comment accéder au service du point de vue technique. Bien que les ontologies *ServiceProfile* et *ServiceModel* fournissent une description assez détaillée du service Web, d'autres informations liées à sa spécification technique telles que le format des messages, les protocoles de transmission, la sérialisation ou l'adressage restent indispensables pour pouvoir y accéder. Aussi, le rôle du *Grounding* (liaison) est-il de combler principalement l'écart qui existe entre les modèles sémantiques de description des services Web et ceux syntaxiques proposés pour la même fin (par exemple, le standard WSDL), dans le but de fournir les détails techniques nécessaires. Pour ce faire, l'ontologie *ServiceGrounding* renvoie au fichier WSDL d'un service Web à travers un lien qui le lie à la description sémantique fournie par les autres sous-ontologies.

### 2.5.2. Web Services Modelling Ontology (WSMO)

WSMO (Web Services Modelling Ontology) est une ontologie conçue pour décrire les divers aspects liés aux services Web sémantiques. La dernière version de WSMO existant à ce jour est D3.1. A l'instar de OWL-S, l'objectif de WSMO est de résoudre le problème d'intégration d'applications basées sur les services Web en définissant un environnement homogène WSMF (Web Service Modelling Framework) [Fensel et al., 2002] qui gère les différents processus liés à ce type de composants logiciels

(description, découverte, composition, etc.). Toutefois, l'ontologie WSMO se distingue par la séparation totale entre les éléments impliqués dans la description des services et les mécanismes utilisés pour formuler les requêtes clients. La correspondance entre l'offre et la demande se fait à travers un composant médiateur. Par ailleurs, pour modéliser les services Web, WSMO utilise le langage WSML (Web Services Modelling Language) [Bruijn et al., 2005]. La description qu'il fournit est soutenu par un mécanisme qui s'articule autour de quatre éléments principaux : les ontologies, les médiateurs, les services Web et les buts.

- **WSMO Ontologies :** Ces ontologies fournissent une terminologie de description sémantique des éléments appartenant à des domaines spécifiques. Composées de concepts, relations, fonctions, instances et axiomes, elles permettent de décrire l'ensemble des informations utilisées par les acteurs d'un service Web de façon compréhensible et interprétable par une machine. Pour rendre possible l'utilisation de différentes ontologies, WSMO permet d'importer directement une ontologie dans une autre dans le cas d'absence de conflits, ou par le biais d'un médiateur quand des conflits d'incompatibilité sont relevés.
- **WSMO Web services :** Les services Web WSMO sont décrits en utilisant deux points de vue différents : l'interface et la capacité. La capacité dans ce langage est similaire au modèle (entrées, sorties, pré-conditions et effets) d'OWL-S. Elle renseigne sur les propriétés fonctionnelles du service. L'interface, quant à elle, décrit comment la fonctionnalité du service Web peut être atteinte en spécifiant son comportement (orchestration et chorégraphie) vis-à-vis de ses partenaires. Par ailleurs, un ensemble optionnel de propriétés non-fonctionnelles prédéfinies peut être également associé aux services Web dans WSMO. La figure 2.6 illustre l'exemple du service Web d'achat électronique de livres « *AchatLivre* » décrit dans WSMO au moyen du langage WSML. La mention « *Capability* » dans ce langage décrit les propriétés fonctionnelles à travers les éléments *precondition*, *assumpltion*, *postcondition*, et *effect* et les propriétés non-fonctionnelles à travers l'élément *nonFunctionalProperties*.

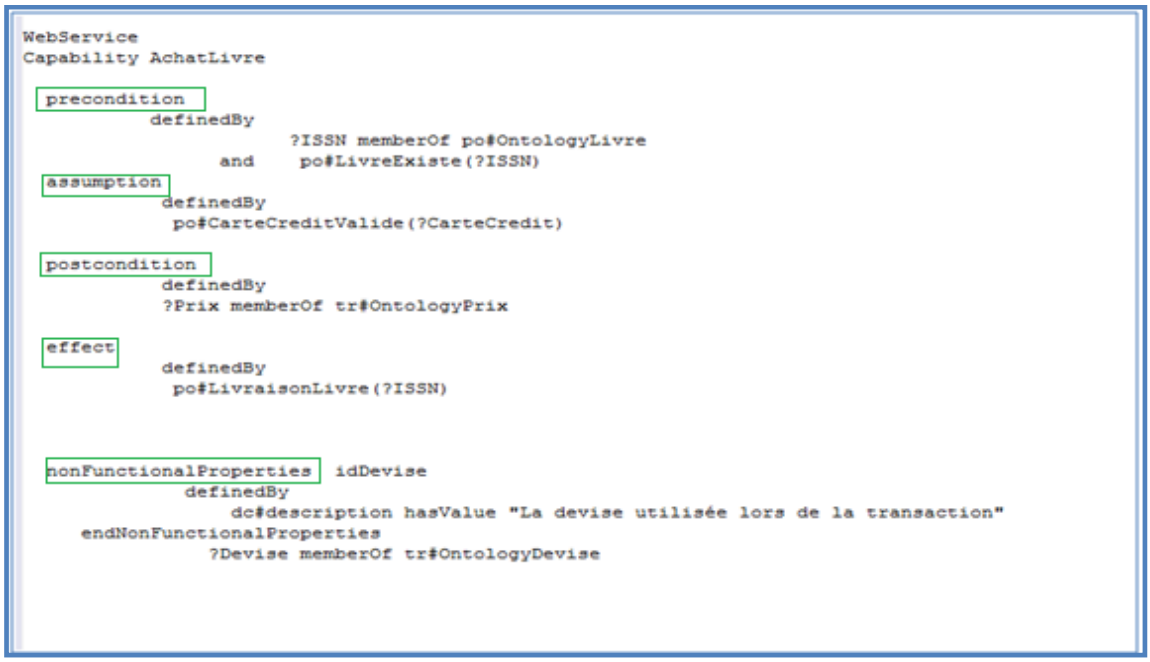


**Figure 2.6**. Extrait du service Web « ***AchatLivre*** » décrit en WSML dans WSMO

- **WSMO Médiators :** Les médiateurs considérés par WSMO permettent de résoudre les incompatibilités structurelles, sémantiques ou conceptuelles qui peuvent être détectées au niveau données ou processus, afin de connecter des ressources WSMO hétérogènes. Dans le cas d'une incompatibilité au niveau données, la médiation sert à établir la correspondance entre les différentes terminologies. Dans le deuxième cas (incompatibilité au niveau processus), la médiation analyse l'exécution des deux services et couvre les éventuelles disparités.
- **WSMO goals :** Le but (goal) dans WSMO définit exactement les objectifs qu'un utilisateur de service Web cherche à satisfaire. Cet utilisateur (qu'il soit un être humain ou un agent logiciel) spécifie ses critères de recherche en décrivant l'interface et les fonctionnalités du service attendues à travers la description du but. Les requêtes clients et les services sont fortement découplées dans WSMO. Les requêtes sont notamment fondées sur les descriptions du but et les médiateurs utilisés doivent chercher des correspondances entre ces requêtes (exprimées au niveau des buts) et les services Web existants.

## 2.6. Analyse des approches de description des services Web

Afin de mieux cerner les caractéristiques de description des services Web, nous proposons une comparaison multi-aspects de l'information descriptive du service Web fournie par les langages et frameworks introduits dans les sections précédentes. Le but de cette comparaison est de relever les propriétés pertinentes et nécessaires qu'un modèle de description de services Web doit comprendre pour qu'il soit d'une part, facile à adopter par la communauté des développeurs des services Web, et d'autre part, suffisamment riche pour favoriser par la suite l'automatisation des différentes activités (découverte, composition, sélection, etc.) qui y sont relatives.

Les couvertures des descriptions fonctionnelle, non-fonctionnelle, de données et technique sont les critères d'évaluation adoptés pour mener cette analyse comparative. Le choix de ces critères relève de la nature des propriétés des services Web dont la classification a été présentée au début de ce chapitre. Nous ne prenons pas en considération les aspects contextuel et comportemental comme critères de comparaison. D'une part, les propriétés contextuelles ne sont traitées explicitement par aucune des approches étudiées. D'autre part, l'aspect comportemental ne sera pas abordé dans les modèles de description et de composabilité des services Web présentés dans les chapitres suivants, puisque nous nous limitons dans le présent travail à la description et la composabilité des services individuels décrits par les standards WSDL 2.0, SAWSDL et WS-Policy 1.5. La description et la composabilté de services composites décrits par les standards WS-BPEL et WS-CDL constitue une perspective intéressante de notre recherche.

### 2.6.1. Couverture de la description fonctionnelle

#### 2.6.1.1. Cas des langages WSDL2.0 et SAWSDL

L'aspect fonctionnel est représenté au niveau de WSDL 2.0 par un ensemble de propriétés fonctionnelles, à savoir l'interface « *Interface* », l'opération « *Operation*», les entrées « *Inputs* » et sorties« *Outputs* » de chaque opération et les messages d'erreurs « *Fault* ». WSDL 2.0 définit ces propriétés au niveau syntaxique seulement. Son extension SAWSDL permet d'associer à chacune de ces propriétés la référence à un concept sémantique, offrant ainsi une description sémantique de l'aspect fonctionnel du

service Web. L'interface est annotée dans SAWSDL par un concept sémantique désignant la catégorie du service. Cette dernière permet d'identifier le domaine fonctionnel du service (par exemple, tourisme, finance, paiement électronique, etc.). L'opération est annotée par sa fonctionnalité (par exemple, réserver un hôtel, effectuer un paiement électronique, etc.). Les entrées, sorties et messages d'erreurs, quant à eux, sont annotés par des concepts sémantiques qui renseignent sur leurs équivalents sémantiques.

#### 2.6.1.2. Cas de l'ontologie OWL-S

Dans le cas de OWL-S, la description fonctionnelle du service Web est définie par le biais du composant « *Service Profile*» qui utilise à son tour, un ensemble de propriétés fonctionnelles. A l'instar de SAWSDL, le service OWL-S est décrit fonctionnellement par sa catégorie « *ServiceCategory* » en utilisant par exemple les classifications fournies par UNSPSC (*United Nations Standard Products and Services Code*) ou par NAICS (*North American Industry Classification System*), sa fonctionnalité « *Service:presents* » et ses entrées « *hasInput* » et sorties « *hasOutput* ».Il peut, en plus, être associé à ses pré-conditions « *hasPrecondition* » et effets « *hasResult* » dans le but de décrire son aspect fonctionnel de manière plus précise en déterminant également les conditions et retombées de son exécution. Par exemple, un service de paiement électronique peut avoir comme pré-condition la validité de la carte bancaire et comme effet la diminution du solde.

#### 2.6.1.3. Cas de l'ontologie WSMO

L'ontologie WSMO fournit un constructeur spécifique au niveau de « *WSMO Web Services Capability »*, appelé capacité, pour décrire les fonctionnalités offertes par un service Web donné. La capacité dans WSMO peut être définie par des pré-conditions *« Precondition »* (dont les entrées qui sont considérées comme un cas particulier des pré-conditions), des assomptions « *Assumptions »*, des post-conditions « *PostCondition* » (dont les sorties, considérées comme cas particulier des post-conditions) et des effets « *Effect* ». En comparaison avec OWL-S, WSMO distingue nettement les post-conditions des effets et les assomptions des pré-conditions. Il utilise les post-conditions pour indiquer les conditions qui assurent le succès de l'exécution du

service, et les effets pour illustrer comment son exécution peut changer l'état du monde. Par exemple, un service de paiement électronique peut avoir comme effet le changement de l'état de la commande de *« en cours »* à *« payée »* et comme post-condition la possibilité de mise à jour des données relatives au solde du client dans la base de données bancaire.

Par ailleurs, WSMO définit un ensemble d'hypothèses (assomptions) qui doivent être satisfaites pour assurer l'exécution du service Web. Ces hypothèses ne sont pas nécessairement vérifiées par le service Web, contrairement aux pré-conditions. Elles explicitent des conditions qui existent dans le monde réel mais qui ne sont pas incluses dans la base de connaissances utilisée par le service. Par exemple, un service de paiement électronique peut avoir comme pré-condition la validité de la carte bancaire et comme assomption le fonctionnement du système bancaire avec qui il doit communiquer et qui ne doit pas être en panne.

#### 2.6.1.4. Discussion

A l'inverse des modèles sémantiques OWL-S et WSMO qui proposent des frameworks de gestion de services Web sémantiques, le standard SAWSDL n'est pas un modèle sémantique en soi. Il constitue un mécanisme d'annotation qui utilise des modèles sémantiques externes. Ce standard dote le langage ancêtre WSDL d'une couche sémantique indépendante des modèles sémantiques pour permettre d'interpréter les éléments qu'il décrit. Aussi, parlons-nous dans le reste de ce paragraphe de l'approche WSDL/SAWSDL qui constitue la vision W3C proposée pour décrire les propriétés fonctionnelles des service Web tout en tenant compte de leur sémantique.

En analysant les caractéristiques des approches WSDL/SAWSDL, OWL-S et SAWSDL en termes de description de l'aspect fonctionnel des services Web, nous avons pu dégager leur degré de couverture de cet aspect et leurs points de convergence ou de divergence. Le tableau 2.1 en fait une synthèse en donnant une vue globale des attributs utilisés par chaque approche dans ce contexte. Notamment, nous constatons que les trois approches décrivent la catégorie métier du service à travers un concept sémantique relié à l'élément interface dans SAWSDL, la propriété « *ServiceCategory* » dans OWL-S et la hiérarchisation des concepts ontologiques au niveau de WSMO.

| Propriété fonctionnelle | WSDL/SAWSDL | OWL-S | WSMO |
|---|---|---|---|
| **Catégorie de service** | Interface/ModelReference | ServiceCategory | Hiérarchie des concepts |
| **Fonctionnalité** | Operation/ModelReference | Service: presents | Web service capability identifier |
| **Entrée** | Input/ModelReference | hasInput | Precondition |
| **Sortie** | Output/ModelReference | hasOutput | Postcondition |
| **Pré-condition** | - | hasPrecondition | Precondition |
| **Post-condition** | - | - | Postcondition |
| **Assomption** | - | - | Assumption |
| **Effets** | - | hasResult | Effect |
| **Messages d'erreur** | Fault, inFault, outFault/ ModelReference | hasPrecondition | Precondition |

Tableau 2.1. Attributs proposés dans la littérature pour la description fonctionnelle des services Web.

Elles décrivent également les fonctionnalités offertes par un service par le biais de concepts sémantiques associés à chacune de ses opérations définies au moyen de l'élément*« Operation»* dans SAWSDL, la propriété « *Service:presents* » dans le cas d'OWL-S et « *Web service Capabilityidentifier»* dans WSMO. A ce stade, nous notons que la notion d'opération n'existe pas dans les modèles sémantiques OWL-S et WSMO. Chaque opération définie au niveau de WSDL 2.0 correspond à un processus atomique dans OWL-S et à un service atomique dans WSMO. De ce fait, un service Web qui offre plusieurs sous-fonctionnalités (opérations) est considéré comme un service composite dont la description des sous-fonctionnalités est fournie au niveau des services (processus atomiques) qui le composent.

Par ailleurs, les messages d'erreurs et leur sémantique renseignés dans WSDL/SAWSDL, au moyen des éléments *« Fault», « inFault», « outFault»* et *« ModelReference»*, peuvent aussi être exprimés sous forme de conditions dans les deux modèles sémantiques OWL-S et WSMO. Les entrées/sorties aussi bien que leur sémantique sont pris en considération par le biais d'éléments réservés (spécifiques) dans le cas de WSDL/SAWSDL et OWL-S, et comme cas particuliers de pré-conditions/post-conditions dans le cas de WSMO. Toutefois, en plus des pré-conditions et effets qui figurent dans OWL-S et WSMO, ce dernier prévoit également la

description des assomptions et post-conditions. Le tableau synthétique 2.1 présente les types de propriétés fonctionnelles couvertes par chacune des trois approches explorées.

### 2.6.2. Couverture de la description de données

Les données échangées entre des services Web doivent être décrites prioritairement par leurs structures de données en vue de permettre leur utilisation ou exploitation ultérieure. Les approches étudiées au niveau de ce chapitre prennent en considération cet aspect. Toutefois, les structures de ces données doivent également être mappées d'un format à un autre pour permettre à un service Web de pouvoir communiquer avec un autre selon un format de données qu'il comprend. Généralement, les services Web communiquent en utilisant des messages XML. Ces messages désignent les données échangées entre ces services.

Dans les modèles sémantiques, les entrées et sorties des services Web sont souvent décrits en utilisant des ontologies. De ce fait, et afin de pouvoir consommer un service Web sémantique par un autre de type client syntaxique, les données sémantiques devraient être traduites dans un schéma XML pour être envoyées au service client. De la même façon, pour pouvoir communiquer avec un service Web sémantique, les données issues d'un modèle sémantique comme l'ontologie devraient être transformés en données XML par le client avant de transmettre son message au service cible. En d'autres termes, les messages sortants d'un service Web sémantique, et inversement, les messages entrants devraient être transformés à partir d'une forme ontologique en schéma XML et vice versa. Dans la suite, nous exposons les manières avec lesquelles les approches étudiées dans ce chapitre assurent la mise en correspondance des données.

#### 2.6.2.1. Cas de WSDL/SAWSDL

Les modèles de données sont représentés au niveau de WSDL par des schémas XML (XSD). Outre l'annotation sémantique d'un élément, SAWSDL permet aussi l'annotation de son type qui peut être simple ou complexe, ou encore l'annotation des attributs de son schéma XML. Cette annotation peut être faite de deux manières *Top-Down* et *Bottom-Up* grâce aux éléments « *LiftingSchemaMapping* » et « *LoweringSchemaMapping* ». L'annotation *Top-Down* associe au type XSD un concept sémantique. L'annotation *Bottom-Up,* quant à elle, associe à chaque élément ou attribut

du type XSD un concept sémantique. Autrement dit, l'élément « *LiftingSchemaMapping* » permet de transformer les données XML d'un message de service Web en format compatible avec un modèle sémantique et l'élément « *LoweringSchemaMapping* » permet de transformer les données d'un modèle sémantique en un message XML. Les deux éléments font référence à des fonctions spécifiques élaborées pour réaliser la transformation concrète de données. Ces fonctions peuvent, à titre d'exemple, être des fonctions XSLT *(Extensible Stylesheet Language Transformations)*. Notamment, ce type de fonctions est généralement adopté pour permettre la transformation des données sémantiques des services Web [Clark, 1999].


**Figure 2.7.** Eléments SAWSDL pour le *mapping* des données

La figure 2.7 illustre l'utilisation des schémas de *mapping LoweringSchemaMapping* et *LiftingSchemaMapping* associés au type *Prix*, définis respectivement par les fonctions de transformation d'URI http://emi/AchatOntology/Ont2PriceType.xml et http://emi/AchatOntology/PriceType2Ont.xml. Ces fonctions permettent de transformer les données de type *Prix* vers le modèle de données du concept d'URI http://emi/AchatOntology/#Prix et vice versa.

#### 2.6.2.2. Cas de OWL-S et WSMO

Les modèles de données, dans le cas de OWL-S et WSMO, sont représentés par des ontologies de domaines en utilisant des langages adaptés à ce contexte. Notamment, pour décrire les structures de données, OWL-S utilise les langages RDF/XML [Beckett et al., 2004] et WSMO opte pour l'utilisation de WSML/XML [Bruijn, 2005]. Par ailleurs, pour décrire le *mapping* de données, WSMO propose un élément spécifique appelé « *Data Grounding* » utilisé pour indiquer le *mapping* des entrées et sorties du

service en schémas XML et OWL-S le fait grâce à « *Service Grounding* ». En vue de réaliser la transformation concrète de données, les modèles utilisés par OWL-S aussi bien que WSMO peuvent être associés à des fonctions de transformation vers des structures XML. Ces fonctions peuvent, à titre d'exemple, être des fonctions XSLT.

#### 2.6.2.3. Discussion

Les trois approches WSDL/SAWSDL, OWL-S et WSMO permettent de représenter les données sémantiques mises en jeu par des services Web. Cependant, nous notons que SAWSDL est un mécanisme d'annotation qui permet de décrire la sémantique des données indépendamment des modèles sémantiques qui peuvent indifféremment être des ontologies, des thésaurus ou tout autre type de formalismes sémantiques. Par contre, la description des données sémantiques par les frameworks OWL-S et WSMO reste dépendante de l'ontologie en question et doit donc être faite conformément à cette ontologie.

Par ailleurs, des informations sur le *mapping* des structures de données des messages XML et des modèles sémantiques associés sont fournis par les trois approches étudiées. Cela permet d'identifier des fonctions de transformation telles que les fonctions XSLT qui réalisent concrètement ce *mapping*. Toutefois, dans le cas de OWL-S et WSMO, le *mapping* proposé ne peut respectivement être fait qu'à partir de ou vers des ontologies OWL-S ou WSMO pour obtenir ou transformer un schéma XML. Par contre, dans le cas de SAWSDL, il serait possible grâce à la nature de l'élément « *ModelReference* », qui peut carrément renseigner sur une liste de concepts sémantiques, de définir plusieurs *mappings* à partir de ou vers des concepts issus de différents modèles sémantiques. Cet avantage de SAWSDL permet au service de s'adapter aux besoins particuliers des clients. Prendre en charge l'opération de *mapping* de modèles sémantiques hétérogènes demeure, en fait, une tâche complexe et coûteuse en temps pour le client.

### 2.6.3. Couverture de la description non-fonctionnelle

Il n'y a nul doute que les propriétés fonctionnelles constituent un noyau pour la description des services Web. Cependant, les propriétés non-fonctionnelles, elles aussi, sont importantes pour déterminer le service le plus approprié aux besoins particuliers d'un client parmi un ensemble de services similaires du point de vue fonctionnel. Dans

cette section, nous exposons la manière dont les approches étudiées au niveau de ce chapitre couvrent ce type de propriétés. Le langage WS-Policy est un standard W3C adapté à la description des propriétés non-fonctionnelles. Tout comme SAWSDL, ses constructions peuvent être incorporées dans un fichier WSDL d'un service Web pour compléter sa description. Aussi, parlons-nous dans le reste de ce paragraphe de l'approche WSDL/WS-Policy.

#### 2.6.3.1. Cas de WSDL/WS-Policy

WS-Policy constitue un formalisme général permettant de décrire les propriétés non-fonctionnelles d'un service Web. Il fournit une grammaire syntaxique simple et extensible permettant de décrire et de communiquer les stratégies d'un service Web. Ce standard est fondé sur XML pour permettre de véhiculer les stratégies dans un mode interopérable. Chaque fournisseur du service peut décrire ses services en utilisant des assertions liées à des domaines non-fonctionnels génériques tels que la sécurité et la qualité de services, ou spécifiques à son contexte et définies selon la syntaxe WS-Policy.

Dans un fichier WSDL 2.0, les politiques peuvent être attachées au *Binding*, à l'ensemble des éléments spécifiés dans le *Binding* (opération, entrée et sortie), aux *Services* WSDL2.0 et *Endpoints*. Toutefois, la description non-fonctionnelle fournie par ce standard reste limitée au niveau syntaxique, et ne couvre pas le niveau sémantique. En plus, l'absence de mécanisme d'annotation pour les propriétés non-fonctionnelles demeure une limite considérable quant à l'utilisation de ce standard.

Par ailleurs, WS-Policy ne spécifie aucun formalisme commun de définition des assertions. Le langage WS-Policy et les WS*- spécifications existantes (par exemple, WS-SecurityPolicy [Nadalin et al., 2009] (Figure 2.4) et WS-Reliability [Iwasa et al., 2004].) ne couvrent à présent que quelques domaines non-fonctionnels spécifiques. De ce fait, les fournisseurs et les clients de services Web sont amenés à définir leurs propres spécifications WS-Policy pour représenter leurs domaines non-fonctionnels spécifiques.

#### 2.6.3.2. Cas de OWL-S et WSMO

Les deux approches sémantiques OWL-S et WSMO utilisent des propriétés spécifiques et précises pour décrire les aspects non-fonctionnels d'un service Web. Notamment, OWL-S prédéfinit quatre propriétés non-fonctionnelles *ServiceName*, *TextDescription*, *ContactInformation* et *QualityRating*. Le paramètre *QualityRating* renseigne sur la classification d'un service dans un système de notation spécifique. Néanmoins, le fournisseur de service a la possibilité, grâce au paramètre *ServiceParameter* défini au niveau du *Service Profile*, d'ajouter autant de nouvelles propriétés non-fonctionnelles qu'il souhaite utiliser pour décrire ses services de manière plus précise.

Par ailleurs, WSMO prédéfinit un ensemble de propriétés non-fonctionnelles, plus riche et varié qu'OWL-S, pour décrire à la fois le service Web et son but. Cet ensemble peut toujours être étendu selon les besoins des fournisseurs de services ou des clients. De plus, les propriétés non-fonctionnelles peuvent être associées à n'importe quel élément utilisé pour décrire un service WSMO. Ceci n'est pas le cas dans OWL-S qui limite l'attachement de propriétés non-fonctionnelles aux éléments utilisés au niveau de l'ontologie *Service Profile*.

#### 2.6.3.3. Discussion

Les trois approches étudiées : WSDL/WS-Policy, OWL-S et WSMO prennent en considération la description des propriétés non-fonctionnelles des services Web. Chacune d'elles couvre cet aspect de manière plus ou moins riche ou étendue. WSDL/WS-Policy et WSMO, en particulier, offrent une certaine souplesse à ce niveau en étendant l'association des propriétés non-fonctionnelles à différents éléments utilisés pour décrire un service Web, et non pas uniquement à ceux relatifs à son aspect fonctionnel. En ce qui concerne le type de propriétés qu'il serait possible d'associer à un service dans ce contexte, l'ensemble des approches évaluées propose des mécanismes pour permettre d'utiliser ou de définir toute sorte de propriété requise. Cependant, cet avantage n'est pas sans inconvénient. Une interprétation sémantique correcte de ces nouvelles propriétés, qui peuvent être définies librement et sans aucun contrôle, n'est pas évidente. Dans ce contexte, le consortium W3C propose d'utiliser des spécifications qui définissent de manière unifiée et standardisée des assertions relatives à un domaine

non-fonctionnel telle que WS-PolicySecurity. Toutefois, jusqu'à nos jours, ces spécifications ne couvrent que quelques domaines spécifiques.

### 2.6.4. Couverture de la description technique

#### 2.6.4.1. Cas de WSDL

L'élément *Binding* utilisé au sein d'un fichier WSDL (Figure 2.8) renseigne sur l'ensemble des détails techniques nécessaires à l'invocation et à la consommation du service Web décrit par ce fichier. Nous rappelons qu'il permet de préciser les protocoles de transport (par exemple, les protocoles SOAP et HTTP), les contraintes techniques telles que les exigences de sécurité, l'encodage, l'adressage et les formats de messages, etc.

```
<!-- Définir les Bindings  -->
  <binding name="InterfaceHttpBinding" interface="tns:AchatLivre"
           type="http://www.w3.org/ns/wsdl/http">
     <operation ref="tns:RechercherParISSN" whttp:method="GET"/>
  ....
  </binding>
```

**Figure 2.8**. Elément *Binding* (Extrait du fichier WSDL 2.0 du service « ***AchatLivre*** »)

Par ailleurs, l'élément *Endpoint* indique une adresse unique URI pour accéder au service (élément WSDL *Service*) et doit être associé à un *Binding* qui, lui aussi, est associé à une interface unique. Nous notons que grâce à ces éléments, WSDL permet, d'une part, d'accéder différemment au même service à travers différents *Endpoints* et différents *Bindings*, et d'autre part, il fournit tous les détails techniques nécessaires pour l'invoquer (Figure 2.9).

```
<!-- Définir les accès au service Web -->
  <service name="RESTfulService" interface="tns:tns:AchatLivre">
     <endpoint name="RESTfulServiceHttpEndpoint"
               binding="tns:RESTfulInterfaceHttpBinding"
               address="http://www.AchatLivre.com"/>
   .....
        </service>
```

**Figure 2.9**. Elément *Service* (Extrait du fichier WSDL 2.0 du service « ***AchatLivre*** »)

#### 2.6.4.2. Cas de OWL-S et WSMO

Au niveau de WSMO et OWL-S, les détails techniques inhérents à l'accès et l'utilisation concrète des services Web ne sont pas explicitement représentés au niveau des ontologies qu'ils spécifient pour les décrire. Notamment, OWL-S utilise le *Service Grounding* pour mapper la spécification abstraite du service à des éléments du fichier WSDL renseignant sur son aspect technique. WSMO, quant à lui, utilise les éléments mécanismes *DataGrounding* et *ChoreographyGrounding* pour assurer cette correspondance.

#### 2.6.4.3. Discussion

L'évaluation de la couverture de l'aspect technique des services Web par les approches WSDL, OWL-S et WSMO a permis de constater que seul WSDL décrit concrètement l'aspect technique d'un service Web. Les approches OWL-S et WSMO s'intéressent plutôt à sa sémantique. Ils visent, en fait, à décrire sémantiquement un service Web abstrait et recourent à son fichier WSDL pour y associer ses détails techniques. Aussi, pouvons-nous conclure que l'utilisation du WSDL pour la description des services Web est inévitable même en cas d'adoption des approches sémantiques OWL-S et WSMO qui recourent finalement aux standards W3C (notamment, le langage WSDL).

## 2.7. Synthèse

Cet état de l'art de la description des services Web nous a permis de tirer des conclusions relatives aux aspects et approches de description des services Web. Il est à noter d'abord qu'une description exhaustive de ces composants logiciels s'avère nécessaire pour favoriser, en particulier, les processus de leur découverte, sélection et composition. Pour être la plus complète possible, elle doit tenir compte de six classes de propriétés de services : fonctionnelles, non-fonctionnelles, contextuelles, orientées données, techniques et comportementales.

Par ailleurs, différentes approches ont été proposées dans la littérature pour décrire les services Web. L'analyse comparative présentée dans la section précédente a exploré les mécanismes de description de services Web offerts, d'un côté, par les standards W3C (WSDL 2.0, SAWSDL et WS-Policy 1.5) et d'un autre côté, par les deux modèles

sémantiques OWL-S et WSMO. A l'issue de cette analyse qui a fait l'objet d'un article de synthèse [Omrana et al., 2013b], nous pouvons noter que :

- les standards W3C se distinguent nettement par leur qualificatif standard et aussi par la simplicité de la description qu'ils proposent. OWL-S et WSMO sveulent procurer de nouvelles normes de description de services Web sémantiques qui peuvent concurrencer le standard WSDL. Certes, ces deux ontologies proposent une richesse de description sémantique notable. Toutefois, elles introduisent une complexité significative et exigent des connaissances pointues pour décrire et gérer les services. Cela explique l'adoption limitée, à ce jour, de ces deux démarches par les industriels et les développeurs de services Web**.**

- l'hétérogénéité des structures des propriétés descriptives utilisées par les deux modèles sémantiques étudiés rend l'appariement d'un service décrit par OWL-S avec un service décrit par WSMO, une opération complexe. Le travail de comparaison que cette opération doit accomplir pour des fins de découverte, composition ou substitution s'avère compliqué vu l'hétérogénéité des structures à comparer. A titre d'exemple, les inputs et outputs dans OWL-S sont considérés respectivement des pré-conditions et post-conditions dans WSMO et les éléments du *Service Profile* de OWL-S diffèrent de ceux définis au niveau de *Web Services Capability* dans WSMO. Par contre, l'utilisation des annotations SAWSDL dans les fichiers WSDL pour faire référence à des concepts sémantiques facilite le processus d'appariement de services, puisqu'il permet d'unifier les structures à comparer.

- A l'inverse de WSDL/SAWSDL, le *mapping* de données proposé par OWL-S et WSMO est également fortement lié à l'ontologie utilisée et ne facilite pas leur réutilisation dans un contexte utilisant une ontologie différente. S'ajoute à cela que la description des services Web proposée par ces deux frameworks reste une description faite à un niveau abstrait et faisant référence à des fichiers WSDL pour se doter des détails techniques nécessaires pour l'accès aux services. Même les propriétés non-fonctionnelles proposées par ces deux frameworks sont associées à des services abstraits, contrairement à la logique flexible de WSDL qui permet d'associer des politiques de WS-Policy à l'implémentation technique du service.

En conclusion, le langage WSDL demeure le standard le plus répandu dans l'industrie des services Web et utilisé pour leur description. Les annotations sémantiques assurées par SAWSDL et les politiques définies par WS-Policy enrichissent et complètent ce langage malgré certaines limites que présente WS-Policy au niveau de la couverture des propriétés non fonctionnelles. Ces standards s'affichent plus avantageux dans la mesure où ils sont compatibles avec WSDL, ce qui facilite leur adoption pour étendre des services existants. Aussi, les mécanismes d'annotation SAWSDL peuvent faire référence à des concepts de OWL-S, WSMO ou autre modèle sémantique.

De ce fait, notre modèle de description des services Web proposé dans le chapitre 4 sera aligné avec les standards W3C à savoir WSDL 2.0, SAWSDL et WS-Policy 1.5 et enrichi pour pouvoir capturer le maximum d'informations relatives aux aspects fonctionnel, non-fonctionnel, inhérent aux données, technique et aussi contextuel. Cette extension est fondée sur les résultats de l'analyse comparative présentée dans le présent chapitre.

# Chapitre 3

# Etat de l'art de la composition des services Web

## Sommaire


***3.1. Introduction*** ***56***

***3.2. Paradigme de composition des services Web*** ***57***

3.2.1 Définitions et cycle de vie de la composition des services Web 57

3.2.2 Propriétés comportementales des services Web composites 60

3.2.1.1. Orchestration 60

3.2.1.2. Chorégraphie 61

3.2.3 Composition statique Vs. composition dynamique 63

***3.3. Découverte des services Web*** ***65***

3.3.1 Spécification de la requête utilisateur 66

3.3.2 Localisation des services Web 67

3.3.3 Appariement des requêtes et des services Web 68

3.3.4 Synthèse 70

***3.4. Construction des plans de composition des services Web*** ***71***

3.4.1 Approches par Workflows 72

3.4.2. Approches par techniques de planification 75

3.4.3. Approches par graphes de dépendances 80

3.4.4. Synthèse 82

***3.5. Discussion*** ***82***

## 3.1. Introduction

Les entreprises sont incessamment exposées à des défis concurrentiels auxquels elles doivent continuellement réagir. En effet, elles ont besoin d'offrir de nouvelles fonctionnalités pour étendre leurs activités commerciales dans des délais optimums. Ces nouvelles fonctionnalités sont souvent construites en réutilisant et agrégeant des services existants tels que les services Web dont les avantages, en termes de modularité, auto-description et interopérabilité, favorisent cette agrégation. Ce besoin de combiner les fonctionnalités offertes par des services Web existants dans le but de créer de nouvelles fonctionnalités métier plus complexes est devenu indispensable. On parle ainsi de la composition des services Web [Alonso et al., 2004].

La composition des services Web constitue un axe de recherche très actif dans le domaine des technologies de l'information vu l'intérêt qu'il présente en termes de réutilisation de composants logiciels. Cependant, le processus de composition s'avère un processus global complexe. Il est composé des processus de découverte, de sélection et de coordination de services qui doivent coopérer pour répondre à un objectif complexe. Plusieurs travaux présentés par la communauté des chercheurs se sont intéressés à ces différents volets de la composition des services Web. Ce chapitre présente un état de l'art de la composition des services Web en vue de discuter de la maturité des recherches et d'identifier d'éventuelles limites auxquelles il faut pallier pour proposer des solutions efficaces tenant compte des besoins et du contexte réels de l'industrie des services Web.

Dans ce qui suit, nous introduisons d'abord le paradigme de composition de services Web en présentant les définitions et les différents types de composition aussi bien que les différentes phases du cycle de vie d'un service composite. Nous exposons ensuite une revue de la littérature qui analyse les mécanismes et approches adoptés pour la réalisation des phases de découverte et de génération de plans de composition qui constituent les deux phases clés de la composition des services Web. Nous concluons finalement ce chapitre par une discussion des approches étudiées.

## 3.2. Paradigme de composition des services Web

La composition de services Web est un processus complexe qui va de la découverte et la sélection de services jusqu'à leur composition proprement dite (leur combinaison) pour répondre à une requête particulière. Cependant, elle n'est toujours pas évoquée dans son sens le plus large et se trouve réduite dans certains travaux de recherche à la phase de combinaison et de coordination des services pouvant coopérer pour assurer ensemble une fonctionnalité complexe. C'est pourquoi nous commençons cette section par la définition de la composition de services Web et la présentation de son cycle de vie. La deuxième partie est consacrée ensuite, à la définition des notions de chorégraphie et d'orchestration de services. Il est également à noter que la manière dont est exécuté le processus de composition des services Web dans la littérature n'est pas figée. Il varie d'un processus statique à un processus dynamique et d'un processus manuel à un autre semi-automatique ou encore automatique. Aussi, la dernière partie de cette section présente-t-elle ces types de compositions et leur contexte d'utilisation.

### 3.2.1 Définitions et cycle de vie de la composition des services Web

Pour répondre à une requête simple, il est souvent question d'invoquer un service Web individuel. Ce service ne possède pas de comportement puisque son exécution se limite à l'exécution d'une opération précise. Par contre, en vue de répondre à une requête complexe, il est nécessaire de composer des opérations d' un ensemble de services Web (Figure 3.1). Cela donne lieu à un service composite qui possède un comportement renseignant sur la manière dont ses composants interagissent lors de son exécution.

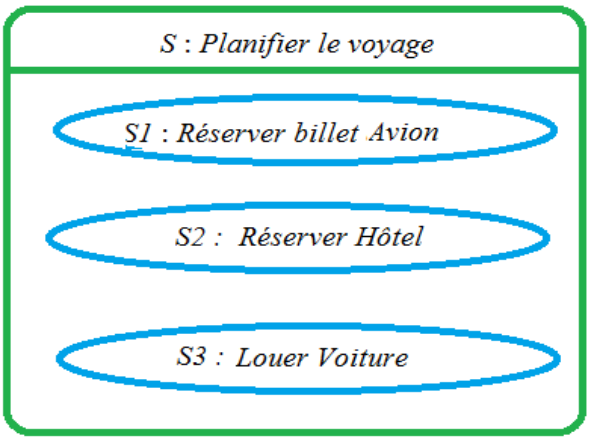


**Figure 3.1** Service Web composite *S* « *Planifier le voyage* »

L'exemple illustré par la figure 3.1 présente le service composite « *S* : *Planifier le voyage* » qui comprend trois services individuels « *S1* : *Réserver billet Avion* », *«S2 : Réserver Hôtel »* et *« S3 : Louer Voiture »*.

La composition d'un service *X* avec un service *Y* se traduit en pratique, par la composition d'une ou plusieurs opérations de *X* avec une ou plusieurs opérations de *Y*. Il est à rappeler que dans le cadre des modèles sémantiques tels que OWL-S et WSMO, la notion d'opération n'existe pas. Chaque opération définie au niveau de WSDL 2.0 correspond à un processus atomique dans OWL-S et à un service atomique dans WSMO (cf. Chapitre 2 pp : 34).

La composition de services Web est notamment un processus complexe qui consiste à trouver des composants appropriés, les sélectionner et les combiner, fournir des informations d'ordre comportemental renseignant sur leur orchestration et chorégraphie en vue de les rendre exécutables. Ces deux derniers concepts d'orchestration et de chorégraphie, servent à décrire le comportement interne et externe d'un service Web composite.

En industrie, le concept de composition des services Web est réduit aux concepts d'orchestration et de chorégraphie. Dans ce contexte, les efforts industriels se sont concentrés sur la description de l'aspect comportemental des services Web composites, en développant des langages de composition spécifiques tels que WS-BPEL et WS-CDL. Ces langages permettent de décrire des plans de composition (traduisant des comportements ou encore la manière dont les composants doivent être agrégés) pour qu'ils puissent, par la suite, être exécutés par des serveurs d'applications tels que Microsoft Biz Talk Server [Microsoft, 2013], IBM WebSphere [IBM, 2007] ou Intalio Server [Intalio, 2013].

Par ailleurs, dans la littérature, les définitions de la composition de services Web couvrent le processus de composition dans son intégralité. Plusieurs définitions ont été proposées. [Benatallah et al., 2005a] considèrent la composition des services Web comme étant un moyen efficace pour créer, exécuter et maintenir des services qui dépendent d'autres services. [Casati et al., 2002], quant à eux, définissent la composition des services Web comme la capacité de constituer de nouveaux services Web composites à valeur ajoutée en combinant des services Web existants et

probablement offerts par différents fournisseurs. [Martin et al., 2004b] la considèrent comme le processus de sélection, de combinaison et d'exécution de services en vue d'accomplir un objectif donné.

En analysant ces différentes définitions, nous pouvons définir la composition de services Web, de manière plus précise, comme un processus qui couvre l'ensemble des activités relatives à la construction de nouveaux services Web exécutables, à partir de services Web existants individuels ou composites ordonnancés, en réponse à un objectif défini.

Comme illustré dans la figure 3.2 schématisant le cycle de vie de la composition des services Web, ces activités consistent à :

1. découvrir les services Web susceptibles de coopérer pour répondre à la requête traitée ;
2. vérifier leur composabilité et générer les plans (solutions) de composition possibles ;
3. sélectionner le ou les meilleurs plans qui répondent aux exigences et préférences du client en termes de propriétés non-fonctionnelles ;
4. exécuter le service composite et le publier.
5. Publier les services composites

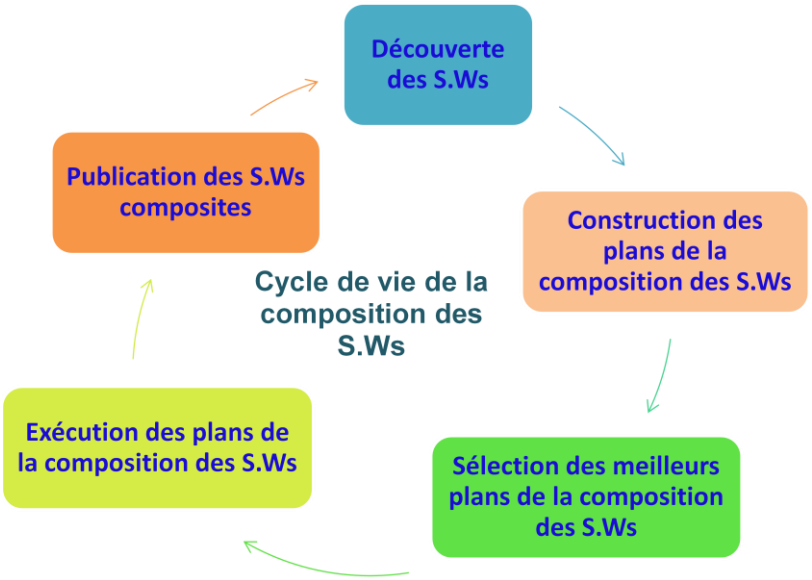


**Figure 3.2** Cycle de vie de la composition des services Web.

La phase de vérification de la composabilité des services Web découverts est souvent réduite, dans les travaux qui se sont intéressés à la composition des services Web (du moins ceux que nous avons étudiés), à un appariement syntaxique ou sémantique des entrées et sorties des opérations à connecter. Toutefois, nous pensons que cette phase constitue une étape cruciale qui permet d'aboutir à des solutions de composition réelles, c'est-à-dire des solutions qu'il est concrètement possible d'exécuter sans erreurs ou problèmes d'incompatibilité entre composants.

### 3.2.2 Propriétés comportementales des services Web composites

Les propriétés comportementales décrivent la manière avec laquelle un service composite est constitué. L'aspect comportemental est traité essentiellement dans le cadre de la composition des services où l'information sur le comportement interne du service composite (la chorégraphie) et le comportement externe des services qui le composent (l'orchestration), est primordiale afin d'interagir avec le service composite. Dans la suite de cette section, nous explicitons en détail ce que les deux concepts de chorégraphie et d'orchestration de services Web désignent.

#### 3.2.1.1. Orchestration

Le comportement externe d'un service composite, appelé orchestration, désigne la manière dont les services invoqués au niveau de ce service, sont agrégés afin de fournir une fonctionnalité plus complexe. L'orchestration permet de décrire l'enchaînement des services selon un canevas prédéfini [Waqar et al., 2004], et de les exécuter comme étant un ensemble d'actions à réaliser par l'intermédiaire de services Web.

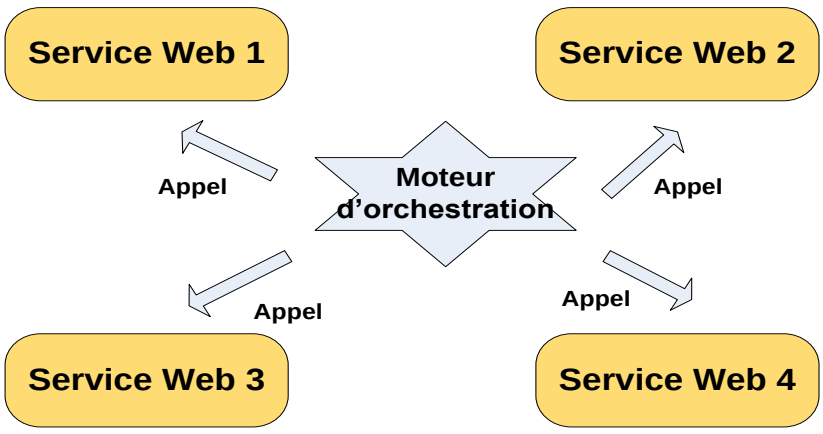


**Figure 3. 3** Vue synthétique de l'orchestration des services Web.

Comme illustré dans la figure 3.3, un moteur d'exécution jouant le rôle de chef d'orchestre, est chargé de gérer l'enchaînement de ces services Web par une logique de contrôle.

Pour la modélisation de l'orchestration des services qui n'est pas couverte par le standard W3C WSDL, OASIS (*Organization for the Advancement of Structured Information Standards*) a conçu et standardisé le langage WS-BPEL [Alves et al., 2007]. Ce langage, fondé sur XML, offre la possibilité de décrire le comportement externe des services Web à un niveau syntaxique. Il spécifie une grammaire XML et un vocabulaire riche pour décrire la logique de contrôle requise pour orchestrer les services Web participant à un processus donné [Wohed et al., 2003]. Ce langage peut être utilisé pour décrire deux types de processus d'orchestration : abstrait et exécutable [Juric, 2006]. Le processus abstrait permet de spécifier les messages échangés entre les différents partenaires sans expliciter leurs comportements internes. Le processus exécutable, lui, précise les différentes opérations de services partenaires, leur ordre d'exécution et l'ensemble des exceptions et des compensations à prévoir.

Dans WS-BPEL, nous pouvons distinguer deux catégories d'activités, primitive et structurée, définies pour décrire l'orchestration des services Web. Les activités primitives, à savoir, *Receive, Reply* ou *Assign*, sont des activités simples utilisées pour décrire comment le processus effectué par le service Web composite doit réagir vis-à-vis des messages échangés. *Receive* marque l'attente de la réception d'un message en bloquant ce processus. *Reply* permet de répondre à un message reçu et *Assign* met à jour les valeurs des variables du processus suite à la réalisation des activités.

Par ailleurs, les activités structurées, telles que *Sequence*, *Switch* ou *Split*, sont des activités complexes combinant plusieurs activités primitives pour décrire différents types de branchement dans un processus.

En outre, le standard WS-BPEL est un langage extensible qu'il est possible d'étendre avec de nouvelles structures en utilisant un espace de nom défini, à condition que ces extensions n'impactent pas la sémantique de ses propres éléments ou attributs.

#### 3.2.1.2. Chorégraphie

Le comportement interne du service, appelé chorégraphie, décrit les différentes interactions (collaborations) que le client de ce service doit respecter afin de consommer

les fonctionnalités de ce dernier. [Barros et al, 2005] définissent la chorégraphie comme un moyen d'atteindre un but commun en utilisant un ensemble de services Web (composants). La collaboration entre chaque service Web de cette collection est décrite par des flots de contrôle. Pratiquement, comme illustré dans la figure 3.4, ces services peuvent collaborer à l'aide de messages échangés afin de savoir si un service peut aider dans l'exécution de la requête.

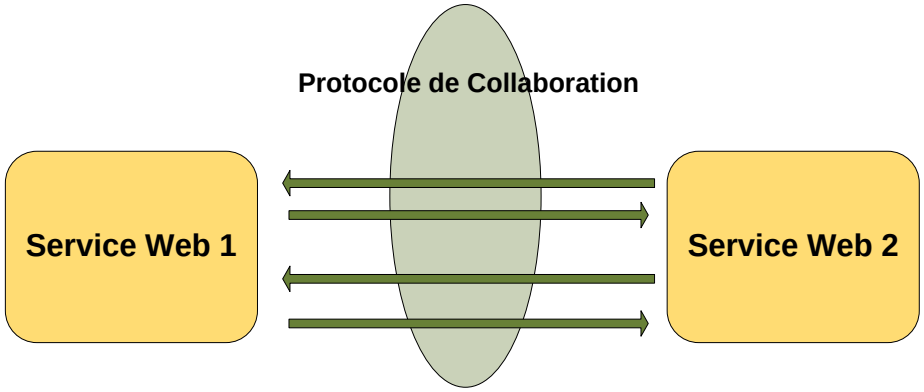


**Figure 3.4** Exemple de chorégraphie de services Web

Pour décrire les interactions possibles entre des services Web et que le standard WSDL ne permet pas de spécifier, le langage WS-CDL (WS-Choreography Definition Language) [Kavantzas et al., 2004 ] a été recommandé par le consortium W3C. Fondée sur XML, tout comme les standards W3C de services Web, la spécification WS-CDL modélise la chorégraphie en décrivant la collaboration point-à-point entre les services Web participants, mais d'un point de vue global [Kavantzas et al., 2004 ].

En effet, WS-CDL complète la description WSDL des services Web par la définition des interactions entre les participants impliqués en décrivant leurs comportements observables. L'exécution de WS-CDL est indépendante des plates-formes et des langages de programmation. Ses principales caractéristiques [Arkin et al., 2002] sont : i) la spécification d'un contrat commun multi-parties qui décrit les comportements des services et assure leur interopérabilité, ii) la gestion des exceptions lors de l'échange des messages entre les services Web et iii) la sécurité qui vise à décrire les échanges de messages entres les services Web.

Un langage de description de chorégraphie des services Web constitue, en fait, un complément aux langages utilisés dans la description du comportement des services.

Dans ce sens, WS-CDL apporte un modèle global qui assure la cohérence des interactions entre des services qui coopèrent. Notamment, la chorégraphie, qu'il décrit, garantit un comportement contractuel entre multiples services sans avoir recours à un outil complexe d'interconnexion. Néanmoins, l'utilisation de ce langage par les industriels reste très limitée, vu qu'il garde encore le statut recommandation et non pas standard de W3C.

### 3.2.3 Composition statique Vs. composition dynamique

Afin de répondre à une requête complexe du client, les services Web disponibles peuvent être combinés pour créer un nouveau service Web composite. Cette combinaison peut être réalisée de façon statique ou dynamique.

Dans l'approche statique, l'agrégation des services Web est réalisée à la phase de conception du service composite [Intalio, 2013]. Les services Web sont déterminés, agrégés et reliés entre eux, pour être par la suite déployés. Ce type de composition est plus approprié aux environnements stables où les services préalablement identifiés par l'utilisateur ne sont pas susceptibles de changer à court terme. Néanmoins, la modification d'un service composant ou sa substitution par un autre suscite également des modifications au niveau de la conception du service composite.

Contrairement à la composition statique où le nombre de services fournis est limité et les services à composer sont spécifiés au préalable, la composition dynamique, elle, est initiée par une requête de l'utilisateur. Elle permet de découvrir, sélectionner et combiner dynamiquement les services à partir des annuaires de services Web, au moment de l'exécution de ladite requête, tout en tenant compte des contraintes de l'utilisateur. Plusieurs approches [(Beauche et al., 2008), (Jiang et al., 2010), (Li et al, 2011)] ont adopté ce type de composition. L'approche dynamique répond à des besoins de flexibilité et d'adaptation. Elle permet en effet de générer des plans de compositions adaptés à chaque requête à partir des services disponibles au moment de la composition.

L'approche dynamique présente plusieurs avantages par rapport à l'approche statique. Outre le fait qu'elle est plus flexible et tient compte des changements qui peuvent survenir aux moments de l'exécution, elle demeure moins restrictive puisqu'elle permet de s'adapter à de nouvelles exigences de services [Shen et al., 2007]. La composition

statique, par contre, considère que les fonctions du service ne changent pas et que les services sont toujours disponibles. Toutefois, la réalisation de la composition dynamique dans un environnement où le nombre de services fournis est en constante évolution n'est pas une simple tâche.

### 3.2.2. Composition manuelle Vs. semi-automatique et automatique

Outre le moment où est réalisée la composition de services Web, et qui la qualifie de statique ou dynamique, le degré d'automatisation peut aussi être pris en considération pour identifier la nature de la composition de services. Selon le critère d'automatisation, celle-ci peut être notamment classée en trois catégories : manuelle, semi-automatique et automatique [Foster et al., 2003].

La composition manuelle est l'approche traditionnelle où l'utilisateur des services Web programme manuellement toutes les étapes du processus de composition des services composites dont il a besoin, en utilisant un éditeur de texte sans l'aide d'outils dédiés. L'inconvénient d'une telle approche est qu'elle exige des connaissances approfondies et des efforts considérables de la part de l'utilisateur qui n'a pas toujours le profil développeur. Cette tâche reste d'une part, ardue et d'autre part, sensible aux erreurs dues au dynamisme et à la flexibilité du Web. Des changements apportés au niveau des services Web composés peuvent affecter le fonctionnement du service composite obtenu et susciter des changements au niveau du processus de composition qui doit être relancé.

Par ailleurs, la composition automatique ou totalement automatisée est une approche qui prend en charge tout le processus de composition et le réalise automatiquement, sans qu'aucune intervention de l'utilisateur ne soit requise. Le processus de composition dans ce cas est un processus générique destiné à différents utilisateurs qui souhaitent chercher et composer des services qui répondent à leurs requêtes spécifiques. Aussi, faut-il que ce processus soit flexible et agile afin de tenir compte des exigences et préférences propres à chaque utilisateur. Ce critère d'agilité s'ajoute aux critères d'exploration d'une multitude de solutions possibles sur le Web et de capacité d'interprétation du fonctionnement des services Web existants, pour faire du processus de composition automatique un processus complexe. La composition automatique est considérée, en fait, comme une tâche d'une grande complexité en raison de la diversité

des éléments (propriétés) des services Web qu'elle a à examiner pour identifier ceux appropriés et compatibles avec les plans de compositions convenables et vérifier leur composabilité. Cette complexité et par la suite le coût de composition augmentent encore à cause de la prolifération rapide des services Web disponibles sur Internet et de la difficulté d'interprétation sémantique et correcte des propriétés de ces services pour répondre aux attentes de l'utilisateur.

Pour pallier des inconvénients du processus de composition automatique des services Web que certains chercheurs dans le domaine considèrent comme trop ambitieux et irréaliste, une troisième approche qui est la composition semi-automatique ou interactive a été proposée. Dans ce genre de composition, l'utilisateur maintient un certain contrôle et supervise le processus de composition envisagé sans avoir besoin de connaissances approfondies en programmation. Il peut, avec l'assistance des outils graphiques, modéliser et concevoir des services Web composites, sélectionner les services appropriés sur la base de suggestions sémantiques proposées par des outils dédiés tels que METEOR-S [Oundhakar et al., 2005], spécifier ses préférences et intervenir en permanence durant le processus de composition.

## 3.3. Découverte des services Web

La découverte des services Web fait référence à l'activité de détection des services Web répondant à une requête donnée. Dans le processus de composition des services Web, la découverte des services est une phase primordiale qui consiste à identifier les services Web susceptibles de coopérer pour former un service composite qui répond à une requête complexe. [Toma et al., 2005] la définissent dans son sens général comme étant le processus qui prend en entrée une requête utilisateur et retourne une liste de ressources ou services pouvant combler éventuellement le besoin décrit. Une autre définition de ce processus a été proposée par [Booth et al., 2004] qui le décrivent comme l'opération de localisation d'une description compréhensible par la machine d'un service éventuellement inconnu au préalable et correspondant à certains critères fonctionnels. Dans la littérature, plusieurs architectures [(Benatallah et al., 2003a), (Benatallah et al., 2005b), (Keller et al., 2005)] ont été proposées pour réaliser la découverte de services Web. Ces architectures traitent la découverte comme un

ensemble d'étapes (Figure 3.5) qui incluent la spécification de la requête utilisateur, la localisation des interfaces de description des services Web et l'appariement (*matching* ou recherche des correspondances) des éléments de la requête et des services trouvés, comme étapes élémentaires dans le processus de découverte.

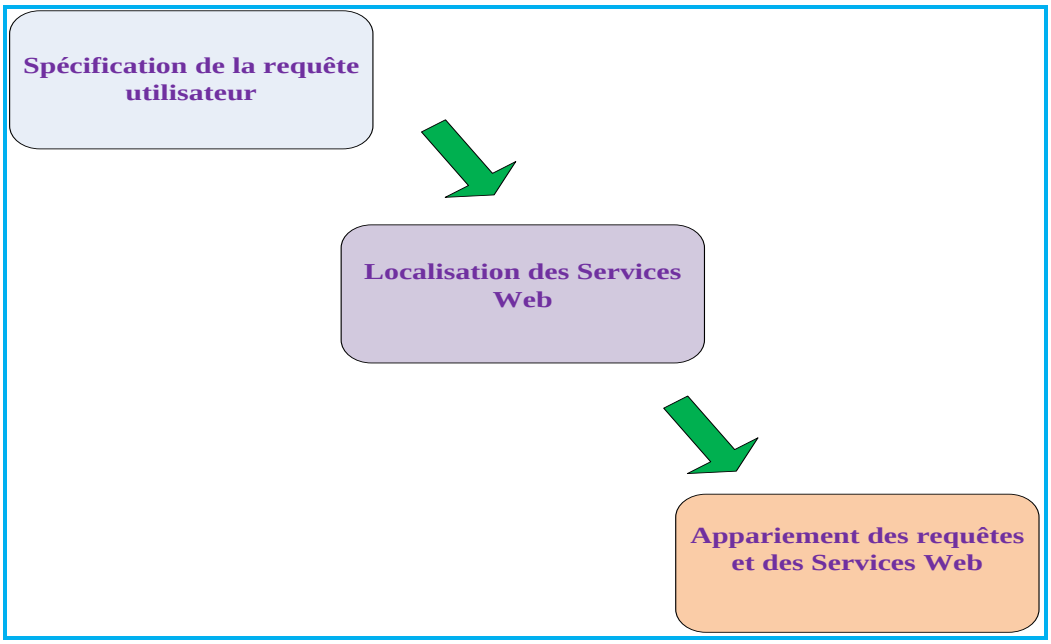


**Figure 3.5.** Principales phases de découverte des services Web

En vue de pouvoir évaluer la maturité des approches de découverte des services Web, nous détaillons dans ce qui suit les approches adoptées dans chacune de ces étapes.

### 3.3.1 Spécification de la requête utilisateur

Les systèmes de composition de services distinguent clairement entre les langages de requêtes et les langages de spécification des services Web. Les langages de requêtes ont pour objectif de permettre aux usagers d'exprimer leurs besoins et exigences d'une manière relativement facile et aisée sans avoir besoin de posséder des connaissances sur les spécifications des services Web. A présent, plusieurs techniques de représentation des requêtes de services Web existent dans la littérature. Nous citons les approches textuelles [(Bosca et al., 2005), (Cremene et al., 2009)], les approches fondées sur la logique [ (Lazovik et al., 2005), (Karakoc et al., 2009)], les approches graphiques [Ceri et al., 1999] ou encore les langages spécifiques aux modèles de services Web

sémantiques (WSML [Bruijn et al., 2005] dans le cas de WSMO et OWL-QL [Fikes et al., 2003] dans le cas de OWL-S

Les approches textuelles utilisent le langage naturel pour exprimer une requête sauf que cela nécessite un traitement postérieur qui consiste à formaliser et réécrire la requête avant de pouvoir la traiter automatiquement. Les approches fondées sur la logique et les graphes utilisent des modèles formels pour décrire la requête. Les approches issues de la logique expriment les requêtes sous forme de prédicats dans le but de pouvoir les traiter en exploitant le potentiel des techniques de l'intelligence artificielle en matière de gestion intelligente des connaissances et surtout de décomposition de requêtes complexes en sous-requêtes simples et ciblées. Les approches fondées sur les graphes expriment les requêtes sous forme de nœuds de départ représentant les entrées du service composite et de nœuds d'arrivée représentant les sorties dudit service. Les nœuds de départ et d'arrivée délimitent le ou les éventuels sous-graphes satisfaisant la requête.

Généralement, la requête utilisateur est décrite en spécifiant les aspects fonctionnels recherchés et les contraintes non-fonctionnelles exigées. L'activité de découverte est effectuée en appariant (*matching*) les propriétés fonctionnelles des services publiés avec la description fonctionnelle de la requête. Les contraintes non-fonctionnelles sont souvent utilisées dans la phase de sélection des services découverts.

### 3.3.2 Localisation des services Web

La localisation des services Web satisfaisant un besoin donné, demeure une étape-clé dans leur découverte. Les approches récurrentes dans ce domaine peuvent être classées en deux catégories, à savoir les approches d'extraction à partir de référentiels centralisés et les approches d'extraction à partir de référentiels distribués.

La première catégorie de ces approches recherchent des interfaces de description des services Web stockées dans un registre (*repository*) central, généralement un annuaire UDDI (*Universal Description Discovery and Integration*) [(Bellwood et al., 2002), (Bellwood et al., 2004)] ou un portail Web, tel que WebServiceList (www.webservicelist.com), WSIndex (www.wsindex.org) ou XMethods (www.xmethods.net). Elles supposent l'enregistrement volontaire des services par leurs

fournisseurs dans les registres en question. Par contre, les approches d'extraction à partir de référentiels décentralisés supposent souvent que les descriptions des services Web sont stockées dans les sites fournisseurs de ces services. Ainsi, la requête utilisateur est exécutée directement au niveau du serveur fournisseur ou par l'intermédiaire d'agents ou robots de collecte des descriptions de services [(Song et al., 2007), (Zhou et al., 2008), (Hong-jie et al., 2008)]. Néanmoins, la collecte et la gestion des informations dans un environnement distribué se présentent comme des tâches fastidieuses et complexes puisque les sites cibles peuvent présenter un problème d'hétérogénéité de présentation et de description des services.

### 3.3.3 Appariement des requêtes et des services Web

L'appariement est une tâche qui vise à évaluer le degré de correspondance entre la requête de l'utilisateur et la description du service Web localisé. Il consiste à relever d'éventuels rapprochement ou similarité qui existent entre les propriétés exprimées au niveau de la requête et celles précisées dans la description du service repéré. Les propriétés fonctionnelles constituent les éléments pertinents auxquels est appliqué l'appariement. Notamment, l'information d'ordre fonctionnel devrait être ciblée en priorité à cette phase de la découverte. Elle est nécessaire pour découvrir les services Web qui peuvent répondre aux besoins fonctionnels de l'utilisateur, en termes de service souhaité, indépendamment de la qualité de ce service et de son degré de réponse à des préférences ou contraintes secondaires de l'utilisateur. Les mécanismes d'appariement utilisés dans ce contexte peuvent être classés en deux catégories majeures, l'appariement syntaxique et l'appariement sémantique dont nous explicitons les principes ci-après :

#### Appariement syntaxique

Les mécanismes d'appariement syntaxique des services Web consistent à mettre en évidence le degré de correspondance entre les structures textuelles utilisées pour décrire les propriétés du service Web et celles exprimées au niveau de la requête utilisateur, en procédant à leur analyse lexicale. Deux techniques d'appariement syntaxique des services Web sont proposées dans la littérature : l'appariement fondé sur le *matching* de mots-clés [Kreger, 2007] et celui fondé sur le *matching* d'interfaces WSDL [Wang et al., 2003]. Le premier consiste à comparer les mots clés de la requête avec ceux qui

décrivent le service Web au niveau de l'annuaire des services. Le deuxième suppose que la requête est décrite au moyen de WSDL sous forme d'un service Web abstrait et réalise le *matching* en comparant les contenus des balises décrivant le service abstrait (service requis) avec ceux des services publiés.

**Appariement sémantique**

Les techniques d'appariement sémantique utilisées dans la découverte de services Web comparent leurs propriétés fonctionnelles avec celles exprimées au niveau de la requête en se référant à leur sens (sémantique). Cette comparaison est réalisée au moyen de deux catégories de méthodes : les méthodes à base d'analyse sémantique latente (Latent Semantic Analysis LSA) [Birukou et al., 2007] et les méthodes fondées sur le calcul de similitudes sémantiques [Zhang et al., 2005].

- ✓ Les méthodes fondées sur l'analyse sémantique latente analysent des documents WSDL en vue d'en extraire la description sémantique des fonctionnalités des services Web publiés. Elles utilisent d'abord des robots de collecte et d'extraction d'information (Information Retreival) ou des moteurs de recherche pour localiser les fichiers WSDL appropriés à partir d'internet [Ren et al., 2008]. L'analyse sémantique latente adoptée par ce type d'approches consiste en général à comparer les mots utilisés pour nommer les éléments de description (interfaces, opérations, paramètres, etc.) dans les fichiers WSDL avec ceux utilisés pour décrire la requêtes, en ayant recours à des dictionnaires lexicaux afin de traiter les synonymes, les mots ayant le même radical, etc.

- ✓ les méthodes de calcul de similitudes sémantiques optent pour le calcul des distances sémantiques entre les propriétés fonctionnelles (entrées, sorties, opérations, etc.) de la requête et des services publiés en vue d'évaluer leur degré de convergence ou de divergence. Ces méthodes se servent principalement d'ontologies de domaine qui fournissent des descriptions sémantiques renseignant sur le sens des concepts utilisés dans le domaine et les relations sémantiques qui les lient [Ji, 2009]. La distance sémantique entre deux concepts ontologiques A et B (appelé aussi nœuds) est calculée sur la base du nombre des concepts (nœuds) qui les séparent et leur profondeur dans la structure

hiérarchique de l'ontologie. Nous traitons plus en détail le calcul des distances entre les concepts dans le chapitre 5. Certaines approches d'appariement sémantique utilisent en plus, des ontologies de contexte [(Spanoudakis et al., 2007) , (Rong et al., 2008) ] qui décrivent des informations relatives au contexte de l'utilisateur et permettent d'intégrer des contraintes de contexte (spécifiées au niveau de la requête) dans le processus d'appariement.

### 3.3.4 Synthèse

La découverte des services Web est essentiellement fondée sur les techniques d'appariement sémantique des éléments de la requête et ceux utilisés pour décrire les services Web publiés. Ces techniques qui utilisent particulièrement les ontologies se sont imposées comme solution pertinente pour pallier plusieurs obstacles de la découverte syntaxique. Toutefois, nous notons que d'autres limitations de la découverte des services Web subsistent toujours, même en ayant recours aux ontologies pour réaliser des appariements sémantiques. Elles sont généralement dues à :

- ✓ **la dépendance de l'ontologie utilisée** : les services concernés par la découverte sémantique fondée sur les ontologies sont souvent des services définis conformément à la structure de l'ontologie correspondante telle qu'OWL-S ou WSMO. Cette contrainte limite considérablement l'éventail des services concernés par le processus de découverte qui se restreint aux services définis par un même type d'ontologie. Dans ce même contexte, l'utilisateur est amené à définir sa requête en utilisant des mécanismes et des règles correspondant à l'ontologie adoptée par le fournisseur pour la description de ses services Web. Cette contrainte se révèle être un obstacle pour un utilisateur non-expert.

- ✓ **l'hétérogénéité syntaxique des structures de données appariées** : les structures de données des paramètres des opérations de services requis par le client peuvent différer de celles utilisées pour représenter les paramètres des services Web publiés. Plusieurs algorithmes d'appariement sémantique [(Benatallah et al., 2005b), (Keller et al., 2005)] ne traitent pas ce problème d'hétérogénéité syntaxique de données appariées. Ils réduisent la comparaison à la sémantique des paramètres, ce qui peut induire à des erreurs lors de l'invocation des services Web découverts.

- ✓ **l'ambiguïté d'expression des contraintes contextuelles** : L'identification des conditions relatives au contexte et aux opérations, quant à elle, nécessite une bonne connaissance du domaine et ne constitue pas une tâche triviale. L'absence d'un langage formel pour exprimer et représenter ce type de contraintes peut conduire à une ambiguïté d'interprétation lors du processus d'appariement.

Dans ce qui suit, nous présentons la phase de construction des plans de composition à partir de services Web découverts, et qui demeure aussi une phase-clé du processus de composition.

## 3.4. Construction des plans de composition des services Web

Dans la composition manuelle et statique, les services Web à composer aussi bien que la manière de les combiner sont déterminés par le concepteur du service composite. A la différence de ce type de composition, la composition automatique et dynamique requiert de trouver d'abord et automatiquement, à la réception d'une requête complexe, des solutions possibles par composition de services, puis de découvrir parmi les services Web existants ceux qui permettent de les concrétiser. Chacune de ces solutions détermine une manière de combiner (composer) un ensemble de services abstraits pour répondre à une requête donnée et constitue un plan de composition dit aussi schéma de composition.

Construire des plans de composition n'est pas une tâche triviale et constitue une problématique traitée par plusieurs travaux de recherche durant cette dernière décennie. Dans la littérature, les techniques investies pour remédier à cette problématique peuvent généralement être regroupées en trois classes d'approches : les approches fondées sur les Workflows (flux de travail) [(Majithia et al., 2004), (Ardagna et al., 2007) , (Fujii et al., 2008)], les approches fondées sur les techniques de planification de l'intelligence artificielle [(Lécué et al., 2007), (Sohrabi et al., 2009a)] et les approches fondées sur les graphes de dépendances [(Ying et al., 2010), (Li et al., 2011)].

Dans cette section, nous présentons et discutons les mécanismes et principes adoptés par chacune de ces approches. Cependant, il est à noter que certains travaux de recherche [(Beauche et al., 2008), (Zeng et al., 2008), (Jiang et al., 2010)] ont également

expérimenté des approches hybrides dans un but d'améliorer l'efficacité des plans produits.

### 3.4.1 Approches par Workflows

En ingénierie des applications, les systèmes de Workflows constituent un moyen de spécification et de composition des activités dans le but de former une chaîne de traitements réalisant un procédé quelconque de gestion. Les techniques de composition des services Web fondées sur les Workflows ont bénéficié des connaissances accumulées par la communauté des chercheurs dans le domaine de gestion des Workflows et de la maturité de leurs approches. Elles considèrent qu'un service Web composite peut être considéré conceptuellement similaire à un Workflow [Casati et al., 2001]. Par analogie à un workflow d'activités, un service composite correspond notamment à un ensemble de services atomiques ou composites interconnectés . De ce fait, l'exécution d'un service Web composite s'avère similaire à l'exécution d'un Worfklow dont les mécanismes assurent la flexibilité, l'adaptation et l'intégration de processus automatiques. La différence réside dans le fait que le Workflow est composé d'activités qui sont remplacées par des services dans le cas de la composition de services Web. Toutefois, ces derniers diffèrent aussi des activités par leur aspect réparti, leur autonomie et leur hétérogénéité [Benatallah et al., 2003b] qu'il faut prendre en considération lors de la gestion ou l'exploitation du Workflow élaboré.

Les frameworks de composition fondés sur la technique de Workflows ont été les premières solutions proposées pour la composition de services Web. Cette technique est souvent appliquée dans le cadre d'une composition statique. La construction des schémas de composition de type statique s'appuie généralement sur des langages de description comportementale pour préciser les interactions entre fournisseurs de services Web et leurs clients. Nous citons à titre d'exemples les langages BPML (*Business Process Modelling Language*) [Intalio, 2002] et WS-BPEL qui sont utilisés pour l'orchestration des services Web aussi bien que WS-CDL [Kavantzas et al., 2004] et WSCI [Arkin et al., 2002] qui sont plutôt utilisés pour spécifier leur chorégraphie. Par ailleurs, nous notons qu'il existe d'autres frameworks utilisant les Workflows pour réaliser des compositions dynamiques tels que Eflow, le framework proposé par Majithia et al. et PAWS (Processes for Adaptive Web Services).

Le framework Eflow [Casati et al., 2000] est une plateforme de spécification et de gestion de services Web composites. Tout service Web composite obtenu est modélisé à un niveau abstrait par un workflow définissant l'ordre d'exécution des différents nœuds du processus métier qui sera exécuté par le service concerné. Ce workflow intègre également les décisions précisant les variantes et les règles de contrôle des flux d'exécution ainsi que les nœuds d'événements. Il est créé manuellement, mais il peut être mis à jour dynamiquement. Lors de la définition d'un nœud abstrait dans Eflow, une requête lui est associée. Une fois que le processus est instancié et que le nœud est activé, la requête s'exécute et retourne une référence à un service Web concret qui sera invoqué. En particulier, la requête s'exécute à chaque activation du nœud afin de remédier au changement fréquent des services dans un environnement dynamique tel que le Web.

[Majithia et al., 2004] proposent un framework qui permet de générer automatiquement à partir d'un objectif de haut niveau renseigné par l'utilisateur, un schéma de composition de services Web. Cette approche se caractérise essentiellement par, d'une part, l'utilisation d'un algorithme de sélection et composition de services dynamique et tolérable aux pannes, et d'autre part, la distinction entre les deux niveaux de granularité de workflows (abstrait et concret), ce qui facilite leur réutilisation et partage. L'architecture du framework proposé intègre plusieurs composants dont principalement le compositeur de workflows abstraits, le générateur de workflows concrets et le référentiel des workflows. Le compositeur de workflows abstraits prend comme paramètre d'entrée un objectif exprimé sous forme de règles par l'utilisateur du framework. Il le transforme en un workflow abstrait représentant un plan de composition qui satisfait l'objectif initial. Le workflow obtenu est constitué par réutilisation de workflows déjà stockés au niveau du référentiel, le cas échéant par exécution d'un algorithme à essais successifs qui permet de trouver une chaîne de services répondant à l'objectif. Le générateur de workflows concrets concrétise ensuite le workflow abstrait en mappant ses nœuds à des services Web existants grâce à un algorithme d'appariement. Si à un nœud donné, aucun service Web n'a pu être mappé, le générateur procède à un appel récursif du compositeur de workflows abstraits qui transformera le nœud en workflow de services abstraits.

PAWS (Processes for Adaptive Web Services) [Ardagna et al., 2007] est un framework développé par Politecnico di Milano pour assurer une exécution flexible et adaptable des services Web composites modélisés sous forme de processus métier. A travers un éditeur standard de BPEL « BPEL Editor » comme illustré dans la figure 3.5, les concepteurs (designers) peuvent définir un processus métier et l'annoter par des contraintes non-fonctionnelles globales et locales liées généralement à l'aspect de qualité de services (QoS). Pour chaque tâche définie par le processus créé, un module intelligent « BPEL Engine » du framework PAWS tente de récupérer parmi les services Web publiés, les services candidats qui correspondent à l'interface requise exprimée en WSDL ou SAWSDL et respectent l'ensemble des contraintes non-fonctionnelles annotées précédemment et exprimées comme des SLA (Service-Level Agreement) [Pereira, 2006]. Dans la plupart des cas, un médiateur « Mediation Engine » est utilisé pour concilier les divergences retrouvées entre l'interface du service Web sélectionné et l'interface requise.

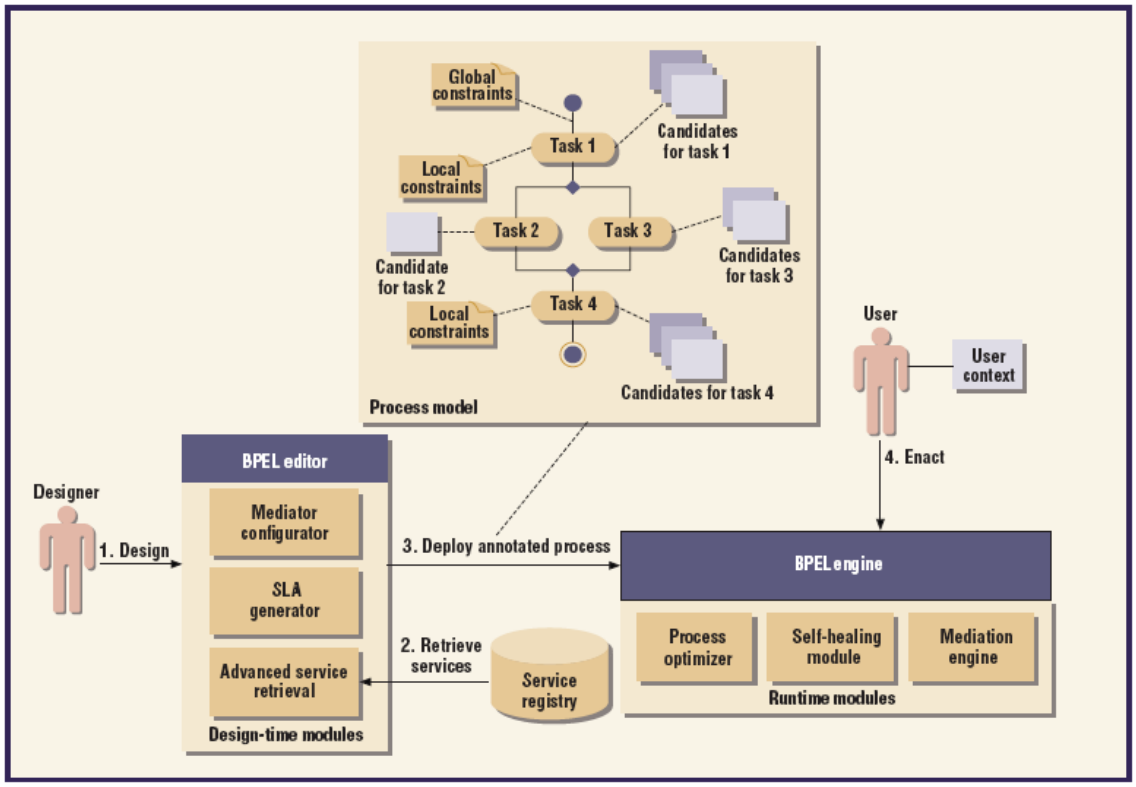


**Figure 3.5.** Composants du Framework PAWS [Ardagna et al., 2007]

Lorsque le processus est finalement exécuté par le moteur BPEL, un service Web candidat est appelé pour chaque tâche. Si au moment de l'exécution, un ou plusieurs

services Web participants échouent, PAWS met en œuvre un ensemble de mesures de recouvrement et de substitution qui assurent l'adaptabilité du processus.

### 3.4.2. Approches par techniques de planification

La planification est une stratégie de l'Intelligence Artificielle (IA), dédiée à la résolution de problèmes. Elle permet de choisir et d'organiser des actions, en fonction d'un but donné. Notamment, cette stratégie considère que la solution (plan) à un problème donné consiste à identifier la séquence d'actions à mener pour passer d'un état initial à un état cible, tout en possédant la connaissance des actions et états possibles aussi bien que des conditions d'application de ces actions.

Aussi, chaque problème à résoudre est-il représenté par un uplet (S, S0, G, A, Γ) où S est l'ensemble des états possibles du monde, S0 est l'ensemble des états initiaux (tel que S0⊂ S), G est l'ensemble des états cibles (tel que G⊂S), A est l'ensemble des actions et Γ est la relation de translation qui spécifie le changement de l'état S1 vers l'état S2 après l'exécution de l'action A (Γ⊆SxAxS).

La plupart des travaux menés pour résoudre la problématique de construction dynamique des plans de composition des services Web utilisent les techniques de planification issues de l'IA. Elles considèrent cette problématique comme un uplet dont les actions représentent des services Web, l'ensemble Γ désigne l'ensemble des pré-conditions et effets des services, l'ensemble S0 indique les conditions initiales et l'ensemble G désigne l'objectif ou la requête client à satisfaire. Aussi, les services Web sont-ils représentés par des actions. Compte tenu d'un objectif donné et d'un ensemble de services Web prédéfini, le planificateur a pour tâche de générer une collection ordonnée de services Web qui appartient à l'ensemble prédéfini et permet d'atteindre l'objectif exprimé.

Plusieurs techniques de planification ont été appliquées au domaine de composition dynamique de services Web : le calcul de situation, le réseau hiérarchique de tâches (HTN), les preuves à théorème et les systèmes à base de règles. L'utilisation de ces techniques est souvent associée à l'utilisation des ontologies (plus précisément d'OWL-S) pour accroître l'efficacité de la composition de services Web à travers la considération de leur aspect sémantique. Dans ce qui suit, nous donnons un aperçu des

principes de chacune de ces techniques tout en faisant référence à certains travaux connexes. Elles sont toutes fondées sur des mécanismes qui ont été explorés en IA pour résoudre des problèmes de planification.

**Calcul de situation**

Le calcul de situation [McIlraith et al., 2002] est un langage logique du premier ordre qui utilise les notions d'actions, de fluents et de situations pour effectuer des raisonnements sur un domaine dynamique (en évolution). Dans ce type de raisonnement, chaque situation *S* est la résultante d'une séquence d'actions appliquée à une situation initiale *S0*. Un domaine dynamique peut être l'objet d'une succession de situations quand il subit un ensemble d'actions diverses. Chaque situation décrit l'historique des actions appliquées à ce domaine sans renseigner sur son état suite à l'exécution de ces actions. Ce dernier est en fait décrit par des fluents qui sont des valeurs de fonctions et de relations appliquées à chacune des situations du domaine.

Golog est un langage de programmation logique de haut niveau conçu à partir du langage formel de calcul de situation en vue de représenter des actions primitives ou complexes dans un domaine dynamique. Ce langage propose un ensemble de constructions extra-logiques utilisées pour définir des actions primitives qui peuvent ensuite être réutilisées pour constituer des actions complexes. Des travaux de la littérature [((McIlraith et al., 2001), (McIlraith et al., 2002)] proposent une approche de composition dynamique des services Web fondée sur le langage Golog. Dans cette approche, la découverte, la sélection et la composition automatiques des services Web sont effectuées par des agents logiciels intelligents. Les auteurs considèrent un service Web comme étant une action qui peut être primitive ou complexe. Les actions primitives ont pour rôle de changer l'état d'un élément, ou encore de changer la connaissance de l'agent. Les actions complexes, quant à elles, sont des compositions d'actions primitives. La base de connaissances de l'agent est en fait constituée de pré-conditions et effets des services Web exprimées au moyen du langage de calcul de situation. L'agent utilise un langage procédural comprenant des constructions telles que si-alors-sinon, quant-fait, etc., combiné à des concepts relatifs aux services Web pour représenter un service composite qu'il génère en répondant à une requête initiale. Cette requête, y compris les contraintes associées (à prendre en considération lors de la

génération du plan de composition), est également exprimée sous forme de procédure générique en utilisant le calcul de situation. L'avantage majeur de l'utilisation des techniques de calcul de situation et plus spécialement le langage Golog est l'expressivité naturelle fournie par ce langage.

Quelques années plus tard, [Phan et al., 2006] conçoivent un système conGolog (extension du Golog) qui permet de mapper les spécifications OWL-S au langage de calcul de situation. Aussi, [((Lécué et al., 2007), (Lécué et al., 2008)] utilisent des techniques de raisonnement logique et les intègrent à une extension du même langage. Récemment, [(Sohrabi et al., 2009a), (Sohrabi et al., 2009b)] ont proposé de tenir compte des préférences utilisateur dans la planification à base de Golog. Les préférences sont exprimées à l'aide d'un langage du premier ordre défini par les auteurs qui utilise une version modifiée de Golog. Les résultats d'évaluation montrent l'efficacité de l'introduction préférences dans la recherche des compositions optimales.

**Réseau hiérarchique des tâches**

La planification au moyen du réseau hiérarchique des tâches (Hierarchical Task Network) est une méthode de planification de l'IA qui crée des plans par décomposition des tâches [Nau et al., 2003]. Les planificateurs HTN différent des planificateurs classiques par la façon dont ils procèdent. L'objectif d'un planificateur HTN est de déterminer une séquence d'actions qui peuvent servir à effectuer une activité ou une tâche. La description d'un domaine comprend un ensemble d'opérateurs semblables à ceux que nous avons notés auparavant et qui servent à décrire un problème de planification. Elle inclut, en outre, un ensemble de méthodes dont chacune est une prescription qui indique comment décomposer une tâche en sous-tâches. Le planificateur de type HTN se sert en fait de ces méthodes pour décomposer les tâches de manière récursive en sous-tâches, jusqu'à l'obtention de tâches primitives qui peuvent être exécutés directement en utilisant les opérateurs de planification. La figure 3.6 présente un exemple de décomposition (HTN) de la tâche « *Construire une maison* », en sous-tâches « *Obtenir le permis de construction* », « *Embaucher un constructeur* », « *Construire* » et « *Payer le constructeur* » et la décomposition de la sous tâche « *Construire* » en tâches primitives « *Construire les fondations* », « *Construire les murs* », « *Construire le toit* » et « *Construire l'intérieur* ».

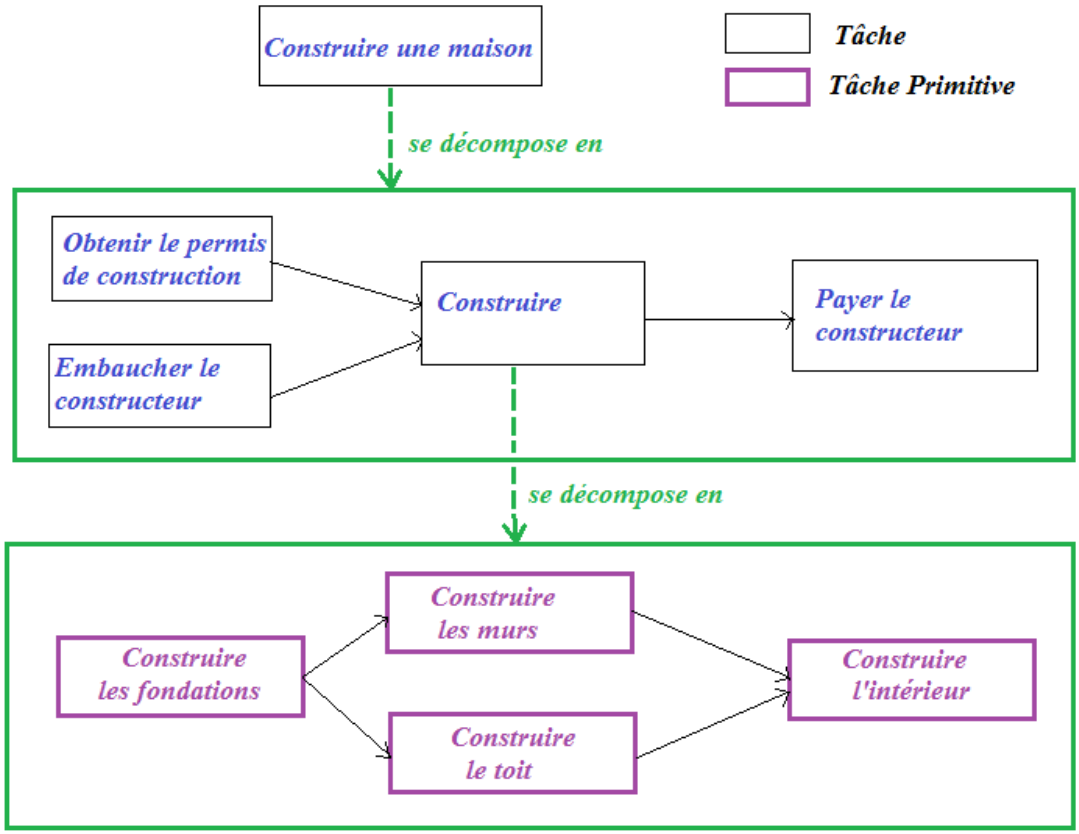


**Figure 3.6.** Exemple de décomposition de tâches (HTN)

Cette démarche s'avère appropriée à la composition dynamique de services Web où les services à composer ne sont pas connus à l'avance ou risquent de changer et doivent être remplacés par d'autres. Quand aucun des services disponibles ne peut concrétiser l'un des services requis, ce service est considéré dans l'approche HTN comme un service complexe qui sera décomposé jusqu'à ce que des services Web primitifs appropriés soient découverts. Le planificateur SHOP2 (Simple Hierarchical Ordered Planner 2) [Sirin et al., 2004] est une implémentation de HTN souvent adoptée dans la littérature. Son avantage majeur est qu'il fournit une solution de planification des tâches proprement dite. Il identifie notamment les tâches mais détermine aussi l'ordre de leur exécution. Aussi, serait-il possible de connaître à chaque étape de la planification l'état actuel du monde étudié.

Un travail plus récent de [Sohrabi et al., 2009a] introduit un algorithme modifié du planificateur HTN (HTNWSC), qui prend en compte à la fois des préférences et des règlements dans la procédure de création du plan de composition.

**Preuves à Théorème**

La composition des services par preuves à théorème est fondée sur la déduction automatique de preuves. La requête de l'utilisateur est décrite comme un théorème que l'on souhaite prouver. Le générateur de preuves de théorème utilise des axiomes ou prédicats qui interprètent des règles de composition, pour générer une ou plusieurs preuves de la requête. Enfin, le plan de composition sélectionné est celui relatif à une preuve particulière dont la concrétisation est possible [Manna et al., 1980].. Le choix de la preuve appropriée est fait en fonction de la disponibilité des services Web.

[(Rao et al., 2003), (Rao et al., 2004)] recommandent l'utilisation des preuves à théorème fondées sur la logique linéaire (LL) pour la composition des services Web. Cette dernière est considérée comme un raffinement de la logique classique qui offre des mécanismes pour décrire des propriétés fonctionnelles et des propriétés non-fonctionnelles qualitatives et quantitatives. L'approche proposée considère deux représentations pour spécifier les services Web, y compris les services Web composites. La première représentation est une description externe utilisant le langage OWL-S. La deuxième représentation est une description formelle des services Web qui les spécifie sous forme d'axiomes de la logique linéaire.

Par ailleurs, [Waldinger et al., 2001] propose de décrire les services Web disponibles et les pré-requis de l'utilisateur selon la logique du premier ordre. Il définit également des axiomes (par exemple, des implications ou des équivalences) qui seront utilisés par le générateur de preuves de théorème pour générer des preuves possibles de la requête.

**Système à base des règles**

Dans les techniques de composition qui optent pour les systèmes à base de règles, les aspects des services Web sont modélisés en mettant l'accent en particulier sur les pré-conditions et effets de leur exécution. Pour chaque service Web, une règle est créée au niveau du système pour préciser que les effets sont atteints si les pré-conditions sont vraies. Un service composite est, en outre, spécifié par un état initial et un état final.

SWORD [Ponnekanti et al., 2001] est un kit de développement qui permet de construire des services Web composites en utilisant des règles pour la génération de plans. Ce kit n'utilise pas les standards de description émergents tels WSDL, OWL-S, WSMO, etc.

mais exploite à la place, le modèle entité-relation (ER). Dans SWORD, les services Web sont modélisés par leurs pré-conditions et leurs post-conditions.

### 3.4.3. Approches par graphes de dépendances

Plusieurs travaux de composition de services Web ont eu recours à la modélisation des dépendances entres les services en utilisant différents types de graphes, tels que les graphes orientés, non orientés, pondérés, acycliques. Tout graphe de dépendance de services Web est constitué d'un ensemble de nœuds et d'arcs qui traduisent des relations inter-nœuds extraites des descriptions de ces services. Selon le formalisme de description adopté (OWL-S, WSDL, WSMO, etc.), les nœuds peuvent représenter différents éléments tels que les services, les opérations, les paramètres, les pré-conditions, les effets, etc. Tout arc liant deux nœuds est créé si leur appariement (syntaxique ou sémantique selon le formalisme de description) détecte leur dépendance (Figure 3.7).

Dans le contexte de composition de services Web, la réponse à une requête complexe consiste à chercher dans le graphe de services Web tous les chemins possibles (solutions possibles) qui à partir des entrées de la requête mènent aux sorties attendues. Ces chemins peuvent notamment passer par différents services et par la suite traduire des plans de composition de services. Des algorithmes de recherche tels que *Forward chaining* , *Backward chaining*, *A**, *Floyd-Warshall* sont également utilisés pour parcourir les graphes et sélectionner le plan le plus approprié en déduisant le chemin optimum suivant des critères prédéfinis. Ces critères peuvent traduire notamment des contraintes et exigences du client, souvent en termes de qualité de service. Par ailleurs, l'approche par graphes de dépendance a été exploitée dans le cas de la composition dynamique des services Web où la construction du graphe est accomplie au moment de l'exécution de la requête du client [(Hashemian et al., 2005), (Gekas et al., 2005)]. Son exécution au préalable de l'exécution des requêtes dans un but d'optimiser le temps de composition est confrontée au problème de changements que subissent les services Web disponibles sur Internet. Ces changements suscitent une mise à jour du graphe à chaque changement, or la mise à jour est en soi une opération complexe et coûteuse en temps.

[Li et al., 2011] utilisent une structure de graphe orienté représentant les dépendances entre les services et leurs entrées et sorties. Ce graphe est construit au préalable avant la

phase de composition dans le but d'optimiser les temps de recherche lors de l'exécution des requêtes clients. Les nœuds représentent des services ou des paramètres (entrées ou sorties). Comme illustré dans la figure 3.6, chaque arc du graphe relie un nœud service « *Si* » à un nœud paramètre « *pi* » et est pondéré en utilisant un poids global de qualité de service calculé auparavant et associé au service en question. Par ailleurs, chaque nœud paramètre peut être lié à une variété de nœuds services. Dans ce travail, la sélection des services, qui peuvent constituer des plans de composition répondant à la requête, est effectuée au moyen des techniques du chaînage arrière « backward chaining » et de la première recherche en profondeur « depth first search », tout en tenant compte de la sémantique des services. Le choix du meilleur plan de composition obtenu est déterminé sur la base de la meilleure qualité de service offerte.

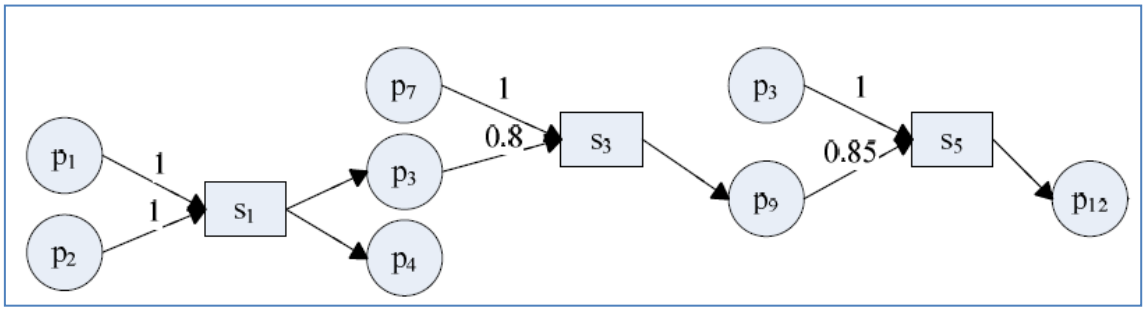


**Figure 3.7**. Exemple de graphe de composition de services Web.

A la différence des recherches de [Li et al., 2011] où la sélection des services pouvant constituer des plans de composition est effectuée au moyen des techniques de chaînage arrière, [Ying et al., 2010] utilise la technique du chaînage avant « *Forward chaining* » pour construire un graphe orienté de composition de services maximal dit« Gmax ». La construction de ce graphe s'initialise à partir des entrées de la requête utilisateur. Le nœud de départ du « Gmax » est le nœud dont les sorties sont équivalentes aux paramètres d'entrée de la requête. Les deux étapes de découverte de services et de matching des entrées et sorties de ces services sont, par la suite, altérés jusqu'à l'obtention des paramètres de sorties de la requête utilisateur parmi ceux des services découverts. Le graphe « Gmax » inclut l'ensemble des sous graphes (plans de composition) qui répondent à la requête client. A partir de ce graphe maximum, un ou plusieurs graphes minimums « Gmin » peuvent être extraits et proposés comme des plans de composition différents satisfaisant la requête utilisateur. L'extraction d'un

« Gmin » à partir d'un « Gmax » est fondée sur la détection des nœuds partagés (nœuds attachés à plusieurs autres nœuds qui les précèdent) et l'élimination des nœuds redondants.

### 3.4.4. Synthèse

La génération de plans de composition est une phase complexe du processus de composition qui constitue une problématique de la composition dynamique des services Web. Les approches étudiés qui ont été adoptées dans la littérature pour résoudre cette problématique peuvent être regroupées en trois classes : les approches par workflows, les approches par techniques de planification et les approches par graphes de dépendance. Toutes ces approches diffèrent par la vision ou la façon dont elles perçoivent la composition des services Web et utilisent des techniques compatibles avec cette vision. Elles considèrent notamment la composition de services Web comme un problème de gestion de Workflows, de planification ou de recherche de chemin optimal et y appliquent des techniques issues d'ingénierie logicielle, de l'intelligence artificielle ou de la théorie des graphes. Nous pensons qu'elles sont toutes matures et équivalentes si nous considérons leur degré d'adaptation à la résolution de problème de composition des services Web et la maturité prouvée des techniques qu'elles utilisent. Nous constatons que ces approches permettent toutes de générer des plans de composition. Leur limitation majeure réside dans la dépendance des solutions de composition proposées aux modèles sémantiques de description des services Web, ainsi que le niveau de composabilité considéré qui se limite souvent à l'appariement des entrées et sorties des services, et ne traitant pas les différentes facettes de vérification de composabilité entre deux ou plusieurs services ou opérations de service à connecter.

## 3.5. Discussion

Dans le processus de composition des services Web, la découverte des services et la génération de plans de composition peuvent être distinguées comme deux phases cruciales autour desquelles est axée la problématique de composition des services Web. Les approches qui ont été proposées dans la littérature dans le but d'assurer ces deux phases sont diverses et inspirées de domaines similaires.

Nous constatons que ces approches ont atteint un niveau de maturité satisfaisant si nous évaluons leur efficacité pour la composition des services Web. Elles adoptent notamment des techniques et mécanismes qui s'avèrent efficaces et adaptés au problème de composition de services Web. Toutefois, nous notons que ces approches présentent des limites si nous discutons de leur efficience. Leur manque d'efficience est dû principalement à la limitation du contexte dans lequel et à l'angle par lequel elles traitent la composition de services Web.

D'une part, comme nous l'avons explicité dans la synthèse des approches de découverte des services Web, ces approches présentent des limites généralement liées à la restriction au contexte d'un modèle de description donné. Cet aspect conduit notamment à des problèmes de dépendance de l'ontologie ou du modèle de services Web utilisés, d'hétérogénéité syntaxique des données matchées et d'ambiguïté d'expression de contraintes contextuelles.

D'autre part, les approches de construction de plans de composition restreignent souvent l'angle de composition à la composition d'entrées et de sorties de services. De ce fait, les plans de composition générés demeurent des plans abstraits dont la composabilité réelle des services n'a pas été vérifiée. Les travaux de composition étudiés ne tiennent en fait pas compte lors de la composition, d'autres éléments ou propriétés de services telles que les politiques de services Web, les propriétés contextuelles, ou d'autres propriétés techniques comme les protocoles de communication à titre d'exemple, qui peuvent constituer un obstacle pour l'exécution des plans de composition obtenus. En outre, les propriétés non-fonctionnelles sont souvent utilisées lors de la phase de sélection du plan de composition optimal. Or, nous pensons que ces propriétés doivent aussi être traitées avant même la construction des plans de composition pour vérifier la composabilité des services candidats à être connectés. Notamment, des hétérogénéités majeures liées aux propriétés non-fonctionnelles de ces services peuvent rendre leur connexion concrète impossible. A titre d'exemple, citons le cas d'un service qui exige que ses entrées soient cryptées. Ce service peut être connecté à un service dont les sorties ne respectent pas cette exigence.

En conclusion, nous pensons que malgré les efforts investis et la maturité des techniques et mécanismes de composition de service Web, le problème majeur de cette

problématique complexe réside d'une part, dans l'hétérogénéité des modèles adoptés pour la description des services Web et la dépendance des solutions de composition proposées desdits modèles, et d'autre part, dans le niveau de composabilité considéré qui demeure un niveau abstrait mettant l'accent uniquement sur l'appariement des entrées et sorties des services, et ne traitant pas les différentes facettes de vérification de composabilité entre deux ou plusieurs services à connecter.

Dans ce contexte, nous proposons dans les chapitres suivants un modèle de description des services Web conforme aux standards W3C et couvrant leurs différents aspects, qui offre une structure unifiée pour les informations descriptives des services Web indépendamment des modèles sémantiques adoptés et un modèle de composabilité des services Web, fondé sur ce modèle standard et devant permettre de s'assurer de la possibilité d'exécution concrète d'un plan de composition abstrait.

# Chapitre 4

# Modèle UML de description des services Web

## Sommaire


***4.1. Introduction*** ***87***

***4.2. Description des propriétés fonctionnelles des services Web*** ***88***

4.2.1. Modèle de base pour la description abstraite des services Web 88

4.2.2. Modèle d'annotation sémantique des propriétés fonctionnelles 90

4.2.3. Description des propriétés contextuelles et conditions des opérations : limites des standards W3C 92

4.2.4. Modèle proposé de description des propriétés contextuelles des paramètres 94

4.2.4.1. Aperçu des solutions pour la description des propriétés contextuelles 94

4.2.4.2. Solution proposée pour la description des propriétés contextuelles des paramètres des services Web 95

4.2.5. Modèle proposé de description des conditions d'opérations 99

4.2.6. Modèle global de la description fonctionnelle des services Web 101

***4.3. Description des données sémantiques*** ***103***

4.3.1. Mécanisme d'échange de données entre les services Web 103

4.3.2. Modèle de description des données sémantiques 103

***4.4. Description de l'aspect technique des services Web*** ***105***

4.4.1. Mécanisme d'accès aux services Web WSDL 105

4.4.2. Modèle de description des propriétés techniques des services Web 105

***4.5. Description de l'aspect non-fonctionnel des services Web*** ***106***

4.5.1. Composants et exemple d'une politique de services Web ..... 107
4.5.2. Modèle de description des politiques de service Web..... 109
4.5.3. Limites de WS-Policy 1.5 ..... 110
4.5.4. Mécanisme d'annotation sémantique et contextuel d'une assertion ..... 112
***4.6. Synthèse ..... 114***

## 4.1. Introduction

Dans les chapitres précédents, nous avons souligné l'importance d'une description de services Web riche, concise et interprétable par la machine, dans le cycle de consommation de ces services. L'étude comparative des approches de description des services Web (les standards W3C de description, OWL-S et WSMO) nous a permis d'établir la pertinence d'adopter une approche de description alignée avec les standards W3C (WSDL 2.0, SAWSDL et WS-Policy 1.5).

Le premier atout de ce choix est que l'utilisation des standards, largement adoptés par la communauté des services Web, renforce l'accessibilité des services fournis dans l'industrie informatique qui a privilégié les outils et techniques compatibles avec les standards pour le traitement ou l'exploitation des produits logiciels.

Le deuxième atout, la séparation des modèles sémantiques du modèle de description des services Web proprement dite telle qu'elle est proposée dans le cadre d'une description alignée avec les standards W3C, offre aux fournisseurs :

i) une structure commune de description ;
ii) la possibilité d'annoter les services par différents modèles sémantiques au choix.

Ces deux avantages simplifient le processus d'appariement des propriétés descriptives annotées par différents modèles sémantiques. En effet, il est difficile d'établir des correspondances exactes entre les éléments de deux modèles sémantiques différents, vue l'hétérogénéité de leurs structures (par exemple, les entrées et sorties des services sont représentées par *hasInput* et *hasOutput* dans OWL-S et par des pré-conditions dans WSMO).

Dans cette optique, nous proposons une approche de description de services Web. Cette approche de modélisation « *Bottom Up* », élaborée suite à l'étude approfondie des grammaires des trois standards W3C (WSDL 2.0, SAWSDL et WS-Policy 1.5), permet de définir un modèle UML qui enrichit les propriétés descriptives procurées par ces standards, par de nouvelles propriétés, dans le but de capturer le maximum d'informations sur un service Web, tout en restant conforme aux standards [(Omrana et

al., 2010b), (Omrana et al., 2011b)]. Les propriétés proposées permettent d'identifier chaque service Web d'une manière précise et non-ambiguë.

Le modèle proposé est présenté dans la suite par parties, mettant en évidence chacun des aspects de description des services Web. Pour chaque aspect, nous présentons des extensions aux éléments du modèle issus des standards W3C, quand certaines limites se présentent.

## 4.2. Description des propriétés fonctionnelles des services Web

### 4.2.1. Modèle de base pour la description abstraite des services Web

Le standard WSDL fournit aux services Web deux types de description : abstraite et concrète. L'élément interface et ses composants représentent la description abstraite du service. Cet élément, introduit dans la version 2.0 de WSDL, utilise des constructions pour favoriser la réutilisabilité de ses composants. Conformément à cette version, notre modèle de propriétés fonctionnelles comprend des éléments, compatibles avec la spécification WSDL 2.0 [Chinnici et al., 2007], qui forment le noyau de base pour la description abstraite des services Web. Ces éléments décrivent l'aspect fonctionnel d'un service Web à un niveau syntaxique. La figure 4.1 illustre ce modèle UML de base qui comprend les quatre classes principales nécessaires pour décrire des services Web abstraits, à savoir, *Interface*, *InterfaceOperation*, *InterfaceFault*, et *InterfaceParameter*.

La classe *Interface* décrit les séquences de messages que le service Web peut envoyer ou recevoir. Elle est désignée comme une agrégation d'opérations (*InterfaceOperation*) et de messages d'erreurs (*InterfaceFault*). Elle est définie aussi par un nom (*name*). Une interface peut éventuellement étendre une ou plusieurs autres interfaces.

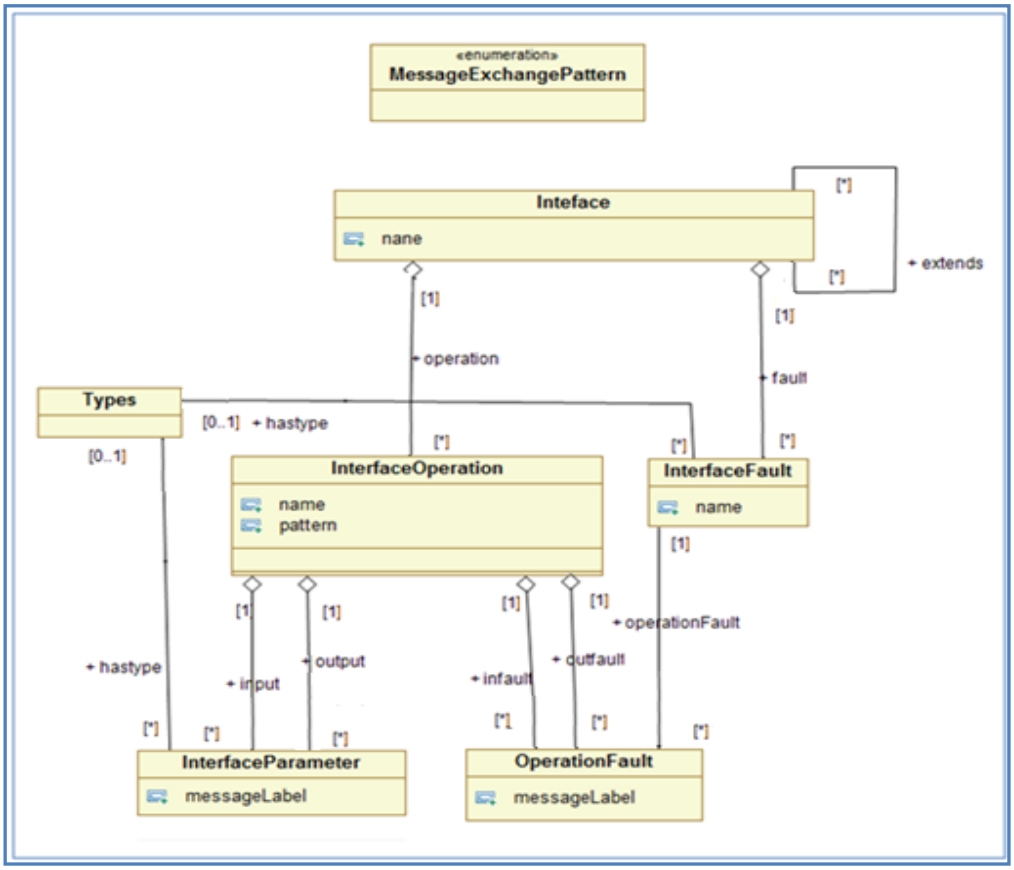


**Figure 4.1.** Modèle abstrait de base des services Web

Le message d'erreur, désigné par la classe *InterfaceFault*, est un événement qui survient pendant l'exécution d'un échange de messages et perturbe son flux normal. Il est utilisé afin de communiquer l'origine de l'erreur. La classe *InterfaceFault* déclare une erreur abstraite en indiquant son nom (*name*) et le type de données (*Types*) associé au contenu du message d'erreur.

L'opération (*InterfaceOperation*) est une interaction du service définissant un ensemble de messages à échanger entre le service et les autres parties impliquées dans l'interaction (y compris les messages d'erreurs). Aussi, la classe *InterfaceOperation* est-elle une agrégation de paramètres d'entrées et sorties (*InterfaceParameter*) et de messages d'erreurs (*OperationFault)* en entrée ou en sortie. Ces messages d'erreurs doivent figurer parmi les messages d'erreurs définis au préalable au niveau de l'interface (*InterfaceFault*). La classe *InterfaceOperation* renseigne, en effet, sur le nom de l'opération (*name*) ainsi que son modèle d'échange de messages (*pattern*) en option. Le modèle d'échange de messages précise le séquencement et la cardinalité des messages échangés lors d'une interaction de l'opération. WSDL prédéfinit huit modèles d'échange

de messages que nous représentons par la classe *MessageExchangePattern* de stéréotype énumération. Parmi ces modèles, nous citons à titre d'exemples le modèle 'In-Only' qui indique que l'opération prévoit un message en entrée seulement et le modèle 'In-Out' qui au contraire, indique que celle-ci prévoit un message en entrée et un autre en sortie. Des nouveaux modèles d'échange de messages peuvent toujours être définis comme prévu par WSDL 2.0.

Par ailleurs, la classe *InterfaceParameter* agrège les différentes entrées et sorties de l'opération. Chaque paramètre a une direction (*messagelabel*) qui indique s'il s'agit d'un paramètre d'entrée (*In*) ou de sortie (*Out*) et un type associé (*Types*).

### 4.2.2. Modèle d'annotation sémantique des propriétés fonctionnelles

Le standard WSDL 2.0 décrit les capacités des services Web au niveau syntaxique sans renseigner sur leur niveau sémantique. Le standard SAWSDL est une extension de WSDL qui permet d'augmenter son expressivité. Il donne la possibilité d'annoter sémantiquement certains éléments décrivant un service Web abstrait, dans le but de rendre leurs capacités interprétables par la machine. Cette annotation est réalisée par le biais du mécanisme *ModelReference*. Dans l'approche SAWSDL, les modèles sémantiques (OWL, OWL-S, WSMO, etc.) qui apportent des informations sémantiques concrètes, sont maintenus à l'extérieur des fichiers d'interface WSDL et sont juste référencés à l'intérieur de ces fichiers au moyen de la balise *ModelReference* proposé par le standard SAWSDL à cette fin. Conformément à ce standard, notre modèle d'annotation des propriétés fonctionnelles des services Web est fondé, comme illustré par la figure 4.2, sur le renvoi vers des modèles sémantiques pour l'annotation de l'interface, des opérations, des messages d'erreurs et des schémas XML de données (XSD) qui décrivent les types des paramètres d'entrée/sortie des opérations.

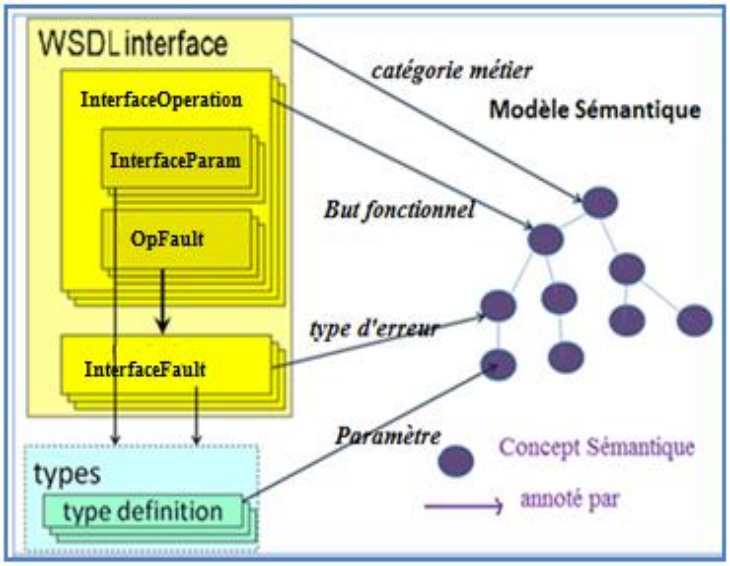


**Figure 4.2.** Modèle d'annotation sémantique des propriétés fonctionnelles des services Web

L'interface d'un service Web peut notamment être annotée sémantiquement par une catégorie métier. Cette dernière permet d'identifier le domaine d'action (éducation, tourisme, etc.) du service en question. L'information sémantique sur la catégorie métier est obtenue à travers le renvoi vers des concepts d'ontologies ou des taxonomies comme NAICS (North American Industry Classification System) et UNSPSC (Universal Standard Products and Services Classification).

Par ailleurs, chaque opération du service peut être annotée par un but fonctionnel qui renseigne sur l'activité réalisée par l'opération (réserver un hôtel, acheter un produit, s'authentifier, etc.). Selon WSDL 2.0, les opérations d'un service Web appartiennent à une interface commune et unique. Par conséquent, ces opérations partagent la même catégorie métier. Aussi, le but fonctionnel fourni par une opération devrait-il être annoté par des concepts sémantiques en concordance avec la catégorie métier du service.

L'annotation sémantique du message d'erreur, quant à elle, fournit une description de haut niveau de l'erreur produite lors de l'exécution d'une opération. A titre d'exemple, l'annotation d'un message d'erreur d'une opération d'achat en ligne par un concept sémantique qui interprète l'indisponibilité du produit permet d'identifier la cause de l'échec de cette opération.

Les types de données des paramètres d'entrées et sorties de l'opération (désignant ses messages d'entrées/sorties) sont souvent définis par des schémas XML de données. Afin de pouvoir les interpréter correctement, il serait nécessaire d'annoter à la fois les éléments, les types et les attributs composant un XSD par des concepts sémantiques.

### 4.2.3. Description des propriétés contextuelles et conditions des opérations : limites des standards W3C

Les services Web sont des composants logiciels sensibles au contexte. Bien que les standards WSDL et SAWSDL offrent un ensemble d'éléments nécessaires pour décrire l'aspect fonctionnel des services Web aussi bien que leur sémantique, ils ne prévoient pas de décrire le contexte d'utilisation fonctionnelle de ces services. Nous mettons en évidence, à travers un exemple, l'incapacité des propriétés fonctionnelles, exposées auparavant, à décrire de manière non-ambigüe la fonctionnalité offerte par un service Web lorsque celle-ci dépend d'un contexte d'usage particulier. Nous y présentons également une brève revue de la littérature mentionnant quelques solutions proposées dans ce cadre.

Considérons le cas de deux opérations A et B (Figure 4.3). Les deux opérations appartiennent à deux services Web d'achat d'ouvrages en ligne, développés par deux entités différentes. Elles sont annotées par le même but fonctionnel « Rechercher Ouvrage » et leurs services appartiennent à la même catégorie métier « Achat en ligne». Chacune prend en entrée un seul paramètre annoté par le concept sémantique « ISSN » et renvoie en sortie deux paramètres annotés par les concepts sémantiques « Intitulé Ouvrage » et « Prix ». Toutefois, bien que ces deux opérations soient similaires du point de vue fonctionnel, leur pertinence en terme de satisfaction des besoins des clients, dépend du contexte de leur utilisation qui est différent.

- ✓ L'opération A qui fournit le prix en Dirham Marocain avec un facteur multiplicateur [1]de 1, et l'intitulé de l'ouvrage en français, est adaptée au contexte marocain.

[1] Un facteur multiplicateur est un nombre utilisé comme multiplicateur pour la mise à l'echelle (traduit de WordNet,http://wordnet.princeton.edu/).

- ✓ L'opération B, qui fournit ce prix en Yuan Chinois avec un facteur multiplicateur de 1000, et l'intitulé de l'ouvrage en chinois, est recommandée dans un contexte chinois.

S'ajoute à cela le fait que l'opération A n'effectue la recherche que sur la base de l'ISSN de la version imprimée de l'ouvrage. Par contre, l'opération B cherche des ouvrages indifféremment de la nature de l'ISSN (version imprimée ou en ligne). La pré-condition associée à l'opération A fait que celle-ci n'est pertinente que lorsqu'elle est utilisée dans le contexte de recherche d'ouvrages imprimés, contrairement à l'opération B.

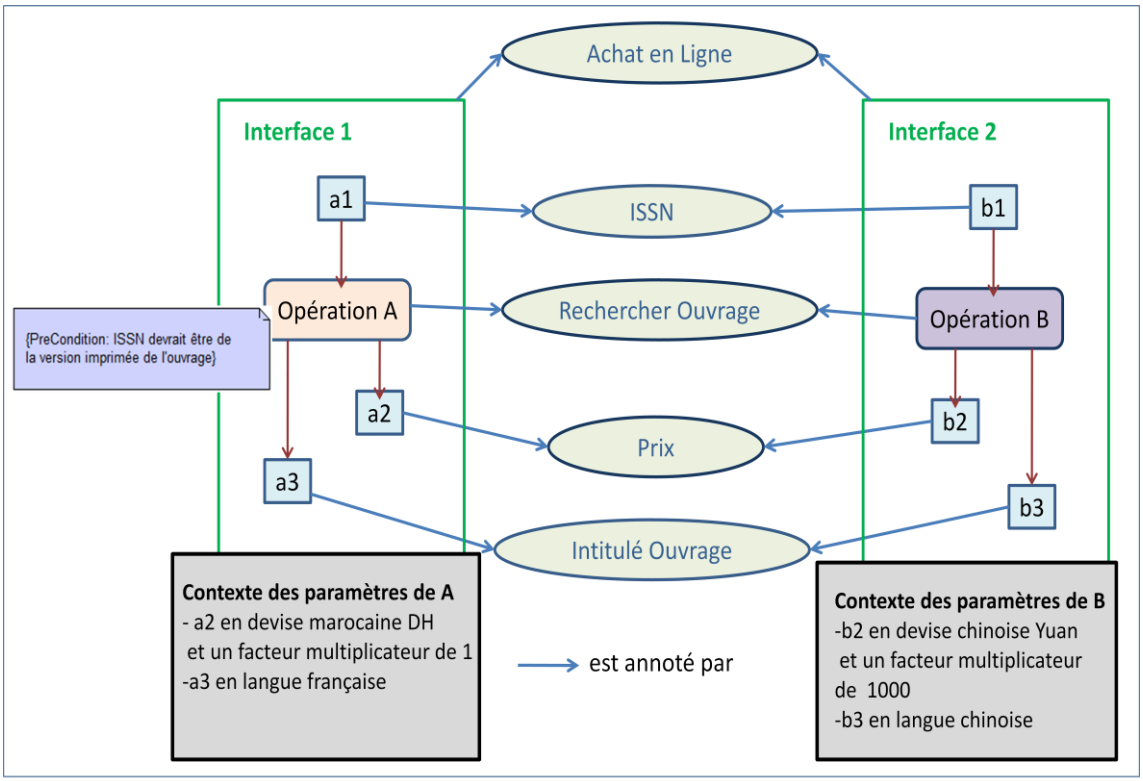


**Figure 4.3**. Exemple de services Web sensibles au contexte

Le service qui impose une pré-condition sur la nature de l'ISSN ne peut être connecté à un service dont les sorties sont des ISSN d'ouvrages en ligne.

En conclusion, la description explicite du contexte fonctionnel des paramètres de l'opération service ainsi que les conditions associées à son exécution, s'avère essentielle surtout pour l'interprétation correcte des données échangées entre deux services et la vérification de leur composabilité (connectivité). Aussi, serait-il nécessaire de proposer

une solution alignée avec les standards WSDL et SAWSDL pour couvrir l'aspect contextuel des services Web.

### 4.2.4. Modèle proposé de description des propriétés contextuelles des paramètres

#### 4.2.4.1. Aperçu des solutions pour la description des propriétés contextuelles

Plusieurs travaux ont abordés la problématique du contexte [(Goh et al., 1999), (Firat, 2003), (Medjahed et al., 2004), (Merrisa et al., 2006a)]. Dans le domaine des bases de données, la notion du contexte a été fortement investie afin de remédier aux hétérogénéités sémantiques des données. Les approches proposées utilisent des ensembles de métadonnées ou des ontologies de contexte pour décrire les aspects sémantiques des données. Dans ce cadre, [Firat, 2003] propose une extension de l'architecture de médiation (intitulée COIN [Goh et al., 1999]) qui fournit une ontologie contextuelle et utilise des mécanismes de médiation en incluant les hétérogénéités ontologiques et temporelles comme partie intégrante de l'architecture proposée. Cette ontologie permet d'associer des attributs sémantiques contextuels aux données. Quant aux mécanismes de médiation, ils spécifient des fonctions de conversion qui établissent des correspondance entre ces attributs contextuels les uns aux autres.

Dans le domaine des services Web, ceux-ci interagissent entre eux et échangent des messages. L'interprétation correcte des données échangées entre des services Web agrégés dépend de la richesse de la représentation sémantique mais aussi de la représentation contextuelle des différents paramètres de ces services et des conditions de leur exécution. Le travail mené par [(Merrisa et al., 2006a), (Merrisa et al., 2006b)] propose un modèle orienté contexte pour représenter la sémantique des paramètres des opérations. Ce modèle utilise des ontologies contextuelles, en plus des ontologies de domaine, pour annoter les éléments de WSDL. Il présente l'avantage d'enrichir les fichiers WSDL par des descriptions contextuelles, tout en étant compatible avec ce standard. Néanmoins, il présente également des limites. Notamment, l'utilisation et la gestion d'ontologies contextuelles couplées à des ontologies de domaine pour couvrir l'aspect contextuel lors de la description sémantique des paramètres se révèlent un peu délicates en pratique. L'annotation des attributs « part » utilisés pour décrire les paramètres de l'opération dans la version WSDL 1.2 [Chinnici et al., 2002], par des

éléments contextuels demeure une description syntaxique qui ne couvre pas la sémantique de ces attributs et n'exploite pas les mécanismes d'annotation de SAWSDL. Enfin, l'enrichissement contextuel proposé couvre uniquement l'aspect fonctionnel des services sans qu'il soit étendu à leur aspect non-fonctionnel.

#### 4.2.4.2. Solution proposée pour la description des propriétés contextuelles des paramètres des services Web

**Illustration des limites**

Comme montré précédemment, la fonctionnalité offerte par une opération d'un service Web donné peut être liée à un contexte d'utilisation spécifique. Aussi, la valeur d'un paramètre d'entrée ou de sortie de cette opération en question doit-elle être traduite selon ce contexte. Chaque paramètre doit être associé à un concept défini dans un modèle sémantique pour avoir une information sur sa sémantique. Toutefois, cette association s'avère, seule, insuffisante pour interpréter d'une manière non ambigüe sa valeur qui reste étroitement liée au contexte d'utilisation du service.

Cette limite sera constatée facilement si nous reconsidérons l'exemple des deux opérations A et B. Ces deux opérations possèdent chacune un paramètre de sortie annoté par le même concept sémantique « Prix ». Cependant, la valeur de ce paramètre, fournie en *MAD* avec un facteur multiplicateur de 1 dans le cas de l'opération A et en *YEN* avec un facteur multiplicateur de 1000 pour B, ne peut être interprétée qu'en tenant compte de ces contextes. A titre d'exemple, des prix de valeurs identiques obtenus par les deux opérations ne peuvent être considérés comme prix identiques sémantiquement puisque leurs unités de calcul sont différentes. Le concept sémantique « Prix » renseigne sur l'équivalent exact du paramètre dans un modèle sémantique qui permet d'avoir des informations sur ses propriétés sémantiques (devise par exemple) et leurs valeurs possibles (Dirham Marocain, Euro, Dollar, etc.). Toutefois, il ne permet pas de déterminer la valeur exacte de la devise considérée pour le calcul du paramètre. Dans l'exemple étudié, la devise et le facteur multiplicateur sont en fait deux propriétés sémantiques nécessaires pour décrire le concept «Prix» soi même et dont l'identification apporte des informations d'ordre contextuel sur le paramètre annoté. De ce fait, nous constatons que l'annotation des paramètres par leurs équivalents sémantiques renseigne sur leurs sens et leurs éventuelles propriétés sémantiques sans déterminer avec précision

leurs propriétés contextuelles. Toutefois, nous notons également que ces propriétés d'ordre contextuel demeurent étroitement liées aux propriétés sémantiques et sont alignées avec celles-ci.

**Modélisation du contexte fonctionnel avec SAWSDL**

Pour assurer la description des propriétés contextuelles des paramètres des opérations des services Web, tout en restant conforme aux standards W3C, à savoir WSDL 2.0 et SAWSDL, notre approche exploite le mécanisme d'annotation sémantique de SAWSDL sans avoir à étendre ces standards par de nouveaux éléments spécifiques à cette fin. Cette approche est fondée sur l'idée qu'un paramètre d'une opération fournie par un service Web doit être annoté non seulement par le concept sémantique équivalent, mais aussi quand cela est nécessaire, par les instances des concepts sémantiques qui décrivent ce dernier et qui caractérisent, en fait, le contexte fonctionnel du paramètre annoté. Par conséquent, nous définissons le contexte fonctionnel d'un paramètre j d'une opération de service i (noté *Contx (*$Param_{ij}$*)* dans la formule ci-après) comme l'ensemble des instances ($Instance_k$) des concepts sémantiques déterminant le contexte de ce paramètre. Ces instances peuvent appartenir à un ou plusieurs modèles sémantiques.

$$\boldsymbol{Contx}(Param_{ij}) = \{Instance_{\boldsymbol{k}} / \ 1 \leq k \leq n \mid n \in N\}$$

L'utilisation d'un modèle sémantique de domaine (une ontologie de domaine par exemple) est nécessaire pour permettre d'annoter les paramètres des opérations d'un service Web par des concepts sémantiques relatifs à son domaine et aussi par des instances de concepts qui fixent leurs contextes fonctionnels. Seulement des modèles sémantiques standards de domaines font défaut. Un modèle sémantique donné peut définir un domaine donné de manière plus ou moins riche par rapport à un autre. Jusqu'à nos jours, il n'y a pas de consensus global sur l'utilisation d'un modèle sémantique de domaine (une ontologie de domaine à titre d'exemple) commun complet et concis. Notamment, deux concepts sémantiquement équivalents appartenant à deux ontologies de domaines peuvent ne pas être définis nécessairement par les mêmes propriétés. Dans un tel cas, le fournisseur du service peut annoter un paramètre par le concept qui lui est équivalent dans un modèle sémantique de domaine, mais aussi par

toute instance d'un autre concept sémantique qui renseigne sur son contexte fonctionnel, même si ce concept émane d'un autre modèle sémantique.

Comme illustré en figure 4.4, le contexte fonctionnel du paramètre *a2* annoté par le concept sémantique *Prix* est fixé par les instances *DH Marocain* et *1*. Ces instances utilisées pour annoter le paramètre *a2* sont liées respectivement aux concepts sémantiques *Devise* et *Facteur Multiplicateur* qui émanent de deux ontologies de domaines différentes. Leur interprétation est déductible à partir de ces ontologies. Par exemple, l'instance *1* serait interprétée comme une valeur du *Facteur Multiplicateur* en se référant à l'ontologie où elle est définie.

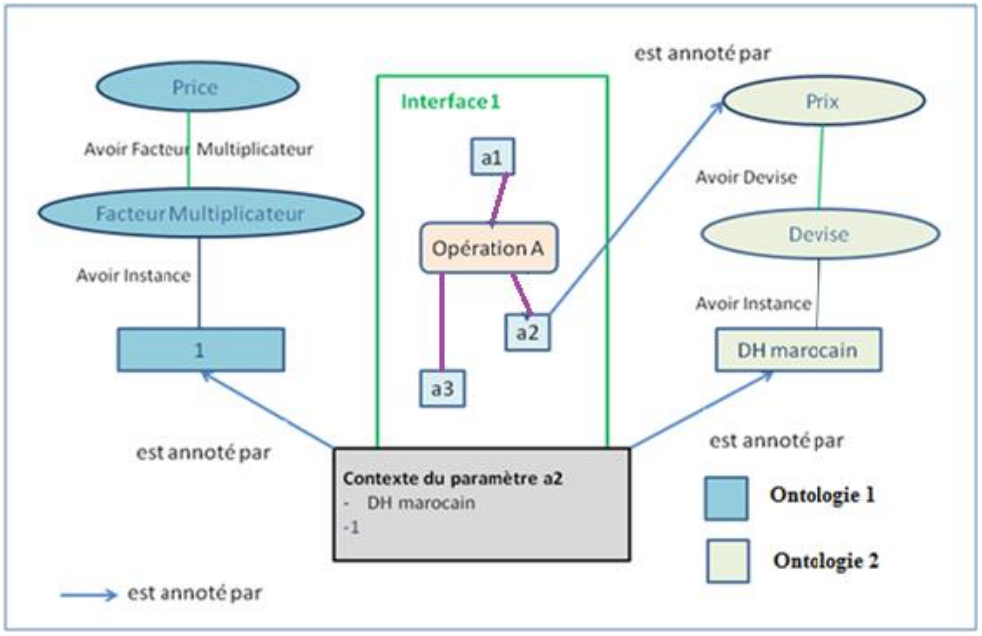


**Figure 4.4.** Exemple d'annotation d'un paramètre dépendant d'un contexte fonctionnel

L'annotation d'un élément donné par des concepts sémantiques qui émanent de différents modèles sémantiques est une opération qui n'est possible que dans le cadre de l'approche SAWSDL. Seul ce standard offre cette flexibilité. Notamment, la balise ModelReference de SAWSDL permet d'associer à un élément WSDL (interface, opération, message d'erreur ou schéma de données) une liste d'éléments appartenant à un ou plusieurs modèles sémantiques. Ces éléments peuvent être des concepts ou des instances. Notre approche exploite cet avantage du standard SAWSDL pour décrire les propriétés fonctionnelles d'un service Web par leurs propriétés contextuelles aussi lorsque celles-ci dépendent d'un contexte donné. Aussi, proposons-nous d'annoter

chaque type XSD de données d'un paramètre d'entrée ou de sortie d'une opération d'un service Web par une liste d'URI d'instances définies dans un ou plusieurs modèles sémantique pour préciser son contexte fonctionnel. Dans la figure 4.5, le paramètre *a2* est annoté au niveau du fichier WSDL par trois URI :

- « *http://Ontology1/Concept#Prix* », l'URI du concept sémantique décrivant le paramètre « *a2*».
- *"http://Ontology1/Concept#Devise#&DHMAROCAIN"* et *"http://Ontology2/nfp#FacteurMultiplicateur#&1"*, les deux URI des instances sémantiques qui informent respectivement sur la devise et le facteur multiplicateur du paramètre « *a2*», pour décrire le contexte fonctionnel de ce paramètre.

```
<xs:element name="a2" type="xs:double"
   sawsdl:modelReference="http://Ontology1/Concept#Prix"
   "http://Ontology1/Concept#Devise#&DHMAROCAIN"
   "http://Ontology2/nfp#FacteurMultiplicateur#&1"   />
```

**Figure 4.5.** Exemple de description sémantique et contextuelle d'un paramètre au niveau du fichier WSDL

Par ailleurs, les propriétés contextuelles d'un paramètre correspondant à un concept sémantique donné ne peuvent pas toujours être prédéfinis et déterminés de manière générique. Certaines propriétés sémantiques d'un paramètre d'une opération ne sont en fait considérées comme propriétés contextuelles que lorsque l'opération concernée dépend d'un contexte donné. Par exemple, la devise est une propriété contextuelle dans le cas de l'opération A qui cible le contexte marocain. Cependant, elle ne peut être considérée comme telle dans le cas d'une opération A1 d'achat en ligne d'ouvrage qui prend en entrée l'ISSN de l'ouvrage et le nom du pays du consommateur du service Web, et retourne en sortie le montant du prix ainsi que sa devise et son facteur multiplicateur. C'est dans se sens que l'annotation des paramètres des opérations en utilisant des ontologies de contexte qui prédéfinissent un ensemble de propriétés contextuelles, peut s'avérer inappropriée. Il revient au fournisseur du service Web de préciser d'une manière concise et complète pour chaque paramètre d'une opération donnée les instances sémantiques qui identifient son contexte fonctionnel.

### 4.2.5. Modèle proposé de description des conditions d'opérations

Les conditions associées à une opération d'un service Web décrivent les contraintes qui régissent son exécution au niveau fonctionnel. Elles spécifient des exigences, des restrictions ou des relations entre les paramètres d'entrées ou de sorties qui doivent être satisfaites au moment ou à la fin de l'exécution de l'opération pour garantir son bon déroulement. Ces contraintes peuvent être de différents types : pré-conditions, post-conditions, assomptions, effets ou autres. Chacun des modèles sémantiques des services Web, tels qu'OWL-S et WSMO, spécifie des types de conditions bien déterminés pour décrire des contraintes sur ces composants logiciels.

**Conditions sur les services OWL-S et WSMO**

Nous notons d'abord, que la notion d'opération est remplacé par service dans les deux paragraphes ci-dessous, vu que cette notion n'est pas géré par les deux modèles sémantiques OWL-S et WSMO (cf. Chapitre 2 pp : 34).

OWL-S propose d'associer des pré-conditions et des effets aux services Web. Dans cette ontologie, les pré-conditions sont des contraintes à vérifier avant l'exécution du service. Par exemple, un service de paiement électronique peut avoir comme pré-condition la contrainte « carte bancaire valide et de type *VISA* ». Les effets, par contre, illustrent les changements qui doivent se produire suite à l'exécution avec succès du service. Par exemple, un effet d'un service de paiement électronique peut être la livraison du produit acheté en ligne au client après son exécution.

WSMO utilise également comme conditions sur les services Web les pré-conditions et les effets, et les enrichit par deux autres types, à savoir, les post-conditions et les assomptions. Les post-conditions sont des contraintes à vérifier après l'exécution du service. Elles informent sur les conditions qui quand elles sont vérifiées confirment le succès de l'exécution du service. Par exemple, après l'exécution d'un service de paiement électronique, la post-condition « le compte du client est débité » est une contrainte qui permet de s'assurer que le service de paiement s'est bien déroulé. WSMO permet également d'associer à une opération un ensemble d'hypothèses (assomptions) qui doivent être satisfaites avant l'exécution du service. Ces hypothèses ne doivent pas nécessairement être vérifiées par le service Web, contrairement aux pré-conditions.

**Mécanismes pour la description sémantique des conditions**

La manière de définir l'ensemble des conditions évoquées ci-dessus dans un modèle sémantique obéit à la même logique suivie pour définir n'importe quel concept sémantique. Cette logique consiste généralement à lier ce concept à un ensemble de concepts et d'instances reliés eux-mêmes par différentes relations sémantiques. Dans le cas de l'ontologie OWL-S à titre d'exemple, la classe OWL *PreCondition* est définie, comme illustré dans la figure 4.6, comme une classe appartenant au champ (*range*) de la classe OWL *Condition*. Toute instance de cette classe (*Condition*) est une formule logique qui peut être évaluée par vrai ou faux. La propriété *hasPrecondition* relie la classe *Process* d'OWL-S avec la classe *Precondition*. Les valeurs de la propriété *hasPrecondition* sont des instances de la classe *Precondition.* Suivant cette logique, toute condition sur une opération peut être définie de manière similaire par des instances d'un concept sémantique décrivant le type de cette condition et proposé par un modèle sémantique donné.

```
<owl:Class rdf:ID="Condition">
<owl:Class rdf:ID="Precondition" />
<owl:ObjectProperty rdf:ID="preCondition">
<rdfs:domain rdf:resource="#Precondition" />
<rdfs:range rdf:resource="#Condition" />
</owl:ObjectProperty>
```

**Figure 4.6.** Définition des classes Condition et Precondition dans l'ontologie OWL-S

**Notre proposition pour annoter les conditions**

Dans ce contexte, nous proposons d'annoter chaque opération d'un service Web par l'URI du concept sémantique décrivant son but fonctionnel, mais aussi par l'ensemble des URI des concepts sémantiques qui traduisent des conditions sur cette opération. Le type de chaque condition pourra être directement déduit du modèle sémantique dans lequel la sémantique de cette condition est définie. Aussi, comme illustré par la figure 4.7, l'opération A est annotée par deux URI qui définissent respectivement son but fonctionnel « *Recherche Ouvrage* » et le contexte de son exécution par une condition sur son paramètre d'entrée *ISSN « ISSNCondition »*.

```
<wsdl:operation name="Operation A" pattern="http://www.w3.org/ns/wsdl/in-out"
      sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/eCommerce#RechercherOuvrage"
                            "http://www.w3.org/2002/ws/sawsdl/spec/ontology/eCommerce#rules#&ISSNCondition"/>
```

**Figure 4.7.** Exemple de description sémantique et contextuelle d'une opération au niveau du fichier WSDL

Par ailleurs, dans la pratique, les conditions associées à l'opération sont souvent complexes et nécessitent des expressions logiques étoffées pour les exprimer. La manière de représenter ces expressions peut être assurée au niveau du modèle sémantique plutôt qu'au niveau du modèle descriptif. Notamment, les modèles sémantiques (OWL-S, WSMO et OWL) utilisent des langages tels que SPARQL [Prud'hommeaux et al., 2008] pour représenter concrètement des expressions formulant des conditions et surmonter leur complexité.

### 4.2.6. Modèle global de la description fonctionnelle des services Web

Les propriétés fonctionnelles d'un service Web renseignent sur ce que peut offrir ce service à ses clients en termes de fonctionnalités. Le standard SAWSDL étend les éléments du modèle abstrait de WSDL 2.0 par des annotations sémantiques pour rendre possible l'interprétation exacte de ces propriétés fonctionnelles. Tout en respectant la logique de ce standard, le modèle que nous proposons pour la description fonctionnelle des services Web [(Omrana et al, 2011a), (Belouadha et al., 2012b)] annote une interface par une catégorie métier, une opération par son but fonctionnel, des messages d'erreurs et des types des paramètres d'entrées et de sortie par des concepts qui leurs sont équivalents sémantiquement. Aussi, ce modèle associe-t-il, les classes *Interface*, *InterfaceOperation*, *InterfaceFault*, et *Types* à la classe *Concept* qui modélise des concepts ontologiques (Figure 4.8).

Par ailleurs, nous exploitons également les mécanismes d'annotation sémantique offerts par SAWSDL pour permettre au modèle proposé de décrire des propriétés contextuelles des entrées/sorties des opérations, ainsi que les conditions associées. Chaque opération propose un nombre bien déterminé de paramètres en entrée et/ou en sortie qui correspondent aux messages entrants et sortants traités par cette opération.

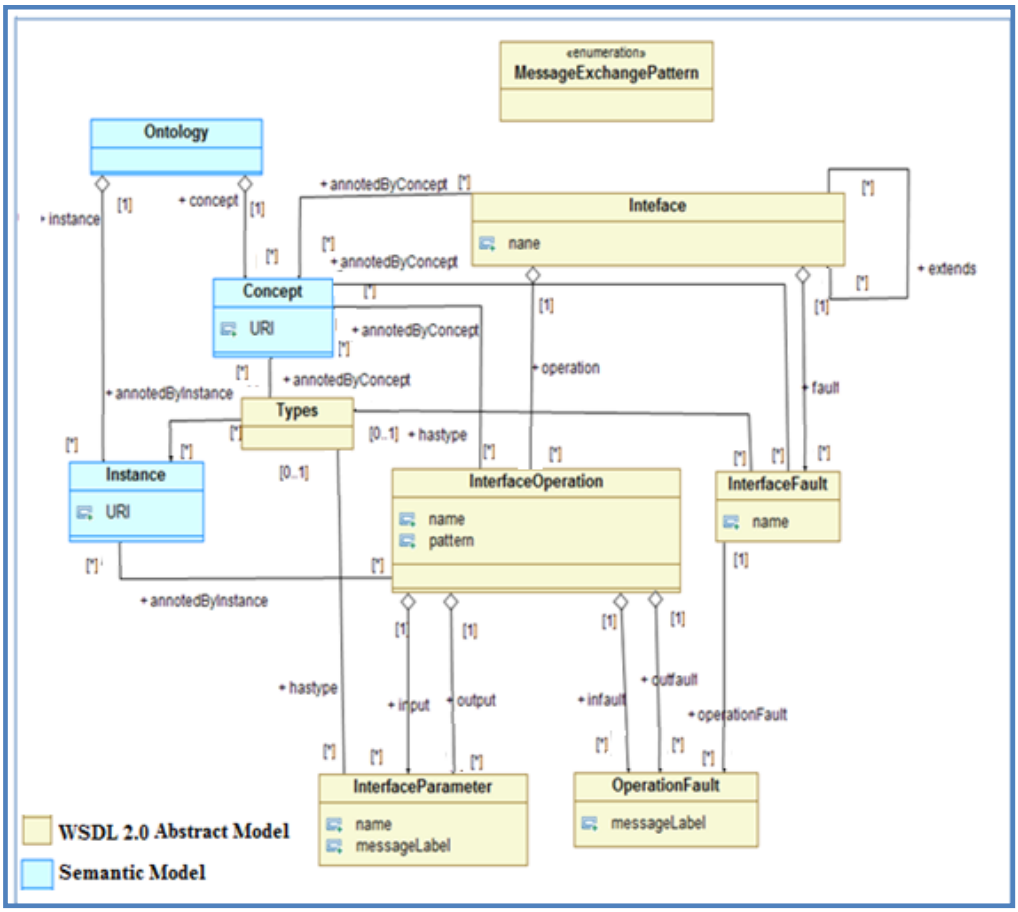


**Figure 4.8.** Modèle global de description des propriétés fonctionnelles

Afin de décrire le contexte fonctionnel de ces messages échangés avec des partenaires ou d'autres services, notre modèle associe la classe *Type* qui renseigne sur le type d'un paramètre à la classe *Instance*. Aussi, chaque paramètre pourrait-il être associé, à travers son type, à un ensemble d'instances qui renseignent sur son contexte fonctionnel et dont la sémantique est définie au niveau d'une ontologie. Dans un fichier WSDL, la liste de ces instances doit être intégrée au niveau du contenu de la balise *ModelReference* associée au modèle de données du paramètre. Aussi, l'exécution d'une opération peut être conditionnée par des pré-conditions à satisfaire, des post-conditions ou des effets à vérifier ou toute autre contrainte liée à son exécution. En plus du concept sémantique décrivant son but fonctionnel, chaque opération d'un service Web doit être annotée également par des instances sémantiques renseignant sur l'ensemble des conditions qui définissent le contexte de son exécution. Aussi, notre modèle associe-t-il les classes *InterfaceOperation* et *Instance*. Dans un fichier WSDL, la liste des instances

sémantiques décrivant des conditions sur une opération doit être introduite au niveau du contenu de la balise *ModelReference* associée à cette opération.

## 4.3. Description des données sémantiques

### 4.3.1. Mécanisme d'échange de données entre les services Web

Les services Web communiquent en échangeant des messages XML qui constituent des entrées ou sorties de ces services. Ces messages sont généralement définis par des schémas XSD. Toutefois, la spécification WSDL 2.0 rend aussi possible l'utilisation d'autres modèles de définition de données tels que DTD (*Document Type Definition*) ou OWL. pour représenter ces messages. Par ailleurs, dans le but de fournir des informations sur la sémantique des services Web, les modèles sémantiques ont souvent recours aux ontologies pour décrire leurs entrées et sorties.

L'hétérogénéité des modèles sémantiques utilisables oblige à traduire les données d'un service Web sémantique dans le format standard XML (Figure 2.7), avant de les communiquer à un autre service Web (client ou serveur). En d'autres termes, les messages sortants d'un service Web sémantique doivent être transformés à partir d'une forme ontologique au format XML, et inversement, ses messages entrants doivent être transformés à partir du format XML dans une forme ontologique donnée. Ces transformations sont généralement assurées par des fonctions de transformation XSLT [Clark, 1999] associées aux données sémantiques devant être communiquées par le service Web.

### 4.3.2. Modèle de description des données sémantiques

La structure d'une donnée est représentée dans le standard WSDL 2.0 par un type souvent décrit par un schéma XSD et annoté selon le standard SAWSDL par un concept sémantique défini dans une ontologie. Afin de rendre possible son échange entre des services Web sémantiques en respectant le mécanisme d'échange de données, le standard SAWSDL permet également d'annoter des éléments de schémas XSD (types simples, complexes, etc.) au moyen de deux constructeurs *LiftingSchemaMapping* et *LoweringSchemaMapping*. L'attribut *LiftingSchemaMapping* indique l'URI d'une fonction qui permet de transformer une structure XML en format d'un modèle

sémantique. L'attribut *LoweringSchemaMapping*, quant à lui, fait référence à l'URI d'une fonction qui transforme les données d'un modèle sémantique en un message XML. Ces deux attributs peuvent notamment référencer une ou plusieurs fonctions de transformation XSLT puisque le mécanisme d'annotation sémantique adopté par SAWSDL permet d'annoter indifféremment les éléments d'un service Web par des concepts sémantiques émanant de diverses ontologies. Dans le cas où ils renvoient vers des URI de plusieurs fonctions de transformation, celles-ci doivent être traitées comme des alternatives à utiliser selon le besoin au moment de l'appariement des données de services Web sémantiques.

Le modèle que nous proposons s'aligne avec les standards WSDL 2.0 et SAWSDL pour la description des données des services Web (Figure 4.9). Il associe la classe *Types* qui modélise des structures des données des services Web aux classes *Concept* et *SchemaMapping* pour informer respectivement sur leurs sémantique et fonctions de transformation (Figure 2.7). Rappelons que cette classe est également associée à la classe *Instance* pour permettre de décrire les propriétés contextuelles des données (paramètres).

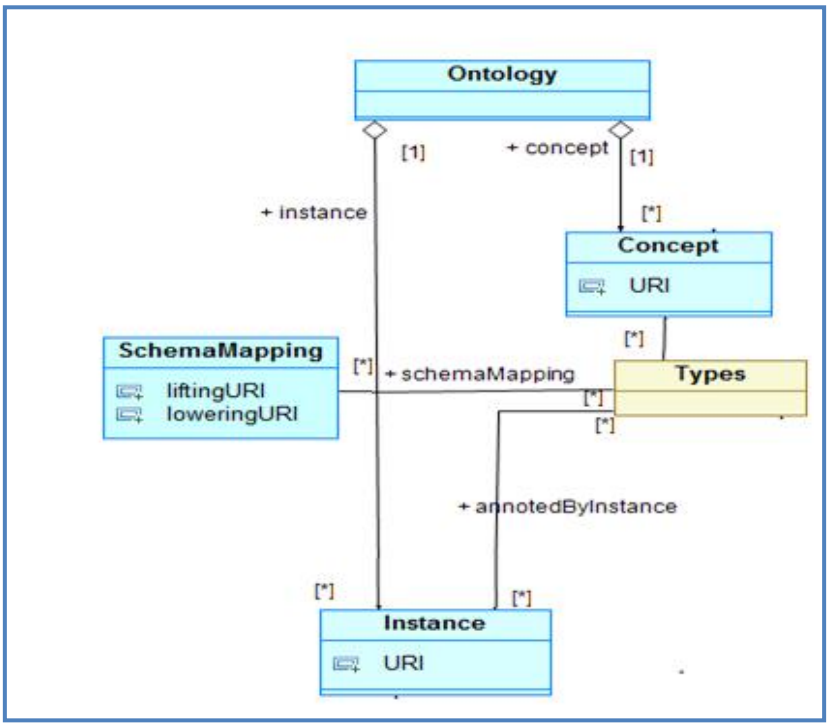


**Figure 4.9.** Modèle de description des données sémantiques

## 4.4. Description de l'aspect technique des services Web

### 4.4.1. Mécanisme d'accès aux services Web WSDL

Le standard WSDL 2.0 adopte un mécanisme flexible et souple d'accès aux services Web. Ce mécanisme permet de rendre un service qui répond, du point de vue fonctionnel, à une seule requête bien déterminée, accessible différemment à travers différents points d'accès et différentes techniques de communication et d'échange de messages. Aussi, WSDL 2.0 distingue-t-il deux niveaux de description des services Web : le niveau abstrait et le niveau concret. Le niveau abstrait décrit ce que fait le service Web (ses propriétés fonctionnelles) alors que le niveau concret précise la manière d'y accéder en décrivant ses propriétés techniques. Ce dernier comprend trois éléments le *Binding*, l'*Endpoint* et le *Service* qui constituent les fondements d'un mécanisme d'accès flexible aux services Web. L'élément *Binding* renseigne sur l'ensemble des détails techniques nécessaires à l'invocation et la consommation d'un service Web. La balise *Endpoint* référence une adresse URI d'un port qui constitue un point d'accès à ce service, tandis que l'élément *Service* énumère l'ensemble de ses points d'accès.

Pour assurer un accès flexible aux services Web, le standard WSDL 2.0 perçoit tout simplement chaque service Web, à travers l'élément *Service*, comme une agrégation d'*Endpoints* qui permettent d'accéder aux fonctionnalités de ce même service avec des liaisons (*Bindings*) différentes auxquelles sont associés différemment ces *Endpoints*.

### 4.4.2. Modèle de description des propriétés techniques des services Web

Etant aligné avec le standard WSDL 2.0, notre modèle de description des propriétés techniques des services Web prévoit trois classes principales : *Service*, *Endpoint* et *Binding* (Figure 4.10). Conformément aux recommandations de ce standard, il spécifie la classe *Service* comme une agrégation d'endpoints et l'associe à la classe Interface avec la contrainte qu'un *Service* n'implémente qu'une et une seule interface. La classe *Endpoint*, quant à elle, est décrite par un nom et un URI, et associée à la classe *Binding* puisque chaque endpoint constitue un port d'accès au service à travers une liaison spécifique.

Par ailleurs, la classe *Binding* est définie par un nom et un type. L'attribut *type* sert à identifier le format de message concret et le protocole de transmission utilisé par le *Binding* (par exemple, SOAP ou HTTP). La classe *TransportProtocol*, stéréotypée énumération, regroupe l'ensemble de protocoles de communication qui peuvent être utilisés.

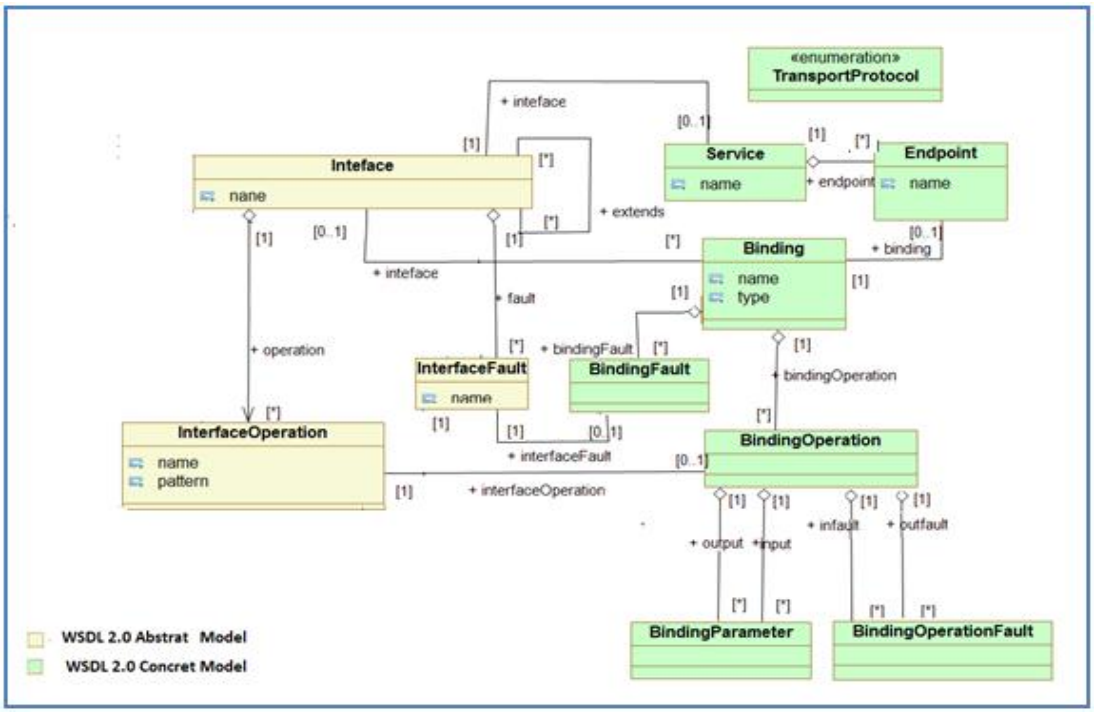


**Figure 4.10.** Modèle de description des propriétés techniques des services Web

La classe *Binding* est associée à la classe *Interface* conformément à WSDL 2.0 qui associe un *Binding* à une seule *Interface* (Figure 2.2). Elle est également perçue comme une agrégation de classes *BindingOperation* et *BindingFault* qui modélisent les liaisons d'opération et des messages d'erreurs de l'interface, et qui par la suite sont associées aux classes *InterfaceOperation* et *InterfaceFault*. La classe *BindingOperation* aggrège elle-même deux classes *BindingParameter* et *BindingOperationFault* modélisant les liaisons des paramètres et des messages d'erreurs des opérations.

## 4.5. Description de l'aspect non-fonctionnel des services Web

A l'instar des propriétés fonctionnelles, les propriétés non fonctionnelles constituent un aspect décisif à prendre en considération lors de la sélection des services Web. Cet aspect, qui couvre des propriétés telles que le temps de réponse, le coût ou la réputation

d'un service Web, représente toutefois un critère déterminant lors de l'évaluation du degré de réponse d'un service aux besoins du client, surtout quand il est question de comparer des propriétés non-fonctionnelles des services à celles d'autres services qui leur sont similaires fonctionnellement. Leur description détaillée est nécessaire pour une découverte, une sélection ou encore une composition efficaces. La modélisation de ces propriétés présente certaines difficultés liées à leur abstraction, à la complexité de leur formalisation, à la diversité et la pluralité de leurs domaines d'application, et surtout à leur dépendance des contextes utilisateurs.

Dans le contexte des services Web, WS-Policy fournit un modèle générique sous forme d'une syntaxe XML proposée pour décrire les politiques des services Web. Ces politiques encapsulent les propriétés non-fonctionnelles de ces composants logiciels sous forme d'assertions appartenant à des domaines non-fonctionnels spécifiques. Dans la suite de cette section, nous proposons un modèle UML aligné avec le standard WS-Policy pour la description des propriétés non-fonctionnelles des services Web [(Belouadha et al., 2010c), (Omrana et al., 2011b)]. Ce modèle est constitué d'un noyau de classes issues de WS-Policy qu'il étend par d'autres éléments permettant de décrire les assertions, aussi bien que leur sémantique et propriétés contextuelles.

### 4.5.1. Composants et exemple d'une politique de services Web

WS-Policy constitue un formalisme recommandé pour décrire les propriétés non-fonctionnelles d'un service Web. Il fournit une grammaire syntaxique simple et extensible permettant de décrire et de communiquer les stratégies des services à travers le Web. Ce standard est basé sur XML pour permettre de véhiculer les stratégies dans un mode interopérable. Chaque fournisseur de service peut définir des assertions liées à des domaines non-fonctionnels génériques tels que la sécurité et la qualité de service, ou spécifiques à son contexte en utilisant la syntaxe proposée par WS-Policy.

Par ailleurs, l'architecture WSDL 2.0 permet d'associer plusieurs implémentations concrètes à une même spécification abstraite du service Web. Les politiques décrites par WS-Policy ne peuvent être attachées qu'aux éléments de la partie concrète (technique) du service Web, à savoir le *Binding* et l'ensemble des ses éléments (opération, entrée, sortie), le *Service* et l'*Endpoint*. Cette logique offre l'avantage de pouvoir séparer la

description fonctionnelle de la description non-fonctionnelle et de pouvoir aussi publier la même fonctionnalité avec différentes contraintes non-fonctionnelles. Les politiques associées aux éléments *Binding*, *Endpoint*, *Service*, *Binding* de l'opération et *Binding* d'une entrée ou sortie, sont appelés respectivement des politiques de niveau *Binding*, *Endpoint*, *Service*, *Operation* et *Message*.

Selon WS-Policy1.5, une politique d'un service Web peut être exprimée par plusieurs politiques alternatives. Chaque politique alternative est constituée d'un ensemble d'assertions. Chaque assertion exprime une condition, une caractéristique ou une exigence (une propriété non-fonctionnelle) associée à un sujet qui peut être un *Endpoint*, un *Binding*, une opération de *Binding*, etc. Les propriétés non-fonctionnelles sont regroupées par domaine selon l'aspect non-fonctionnel concerné. Chaque assertion est définie par un type XSD.

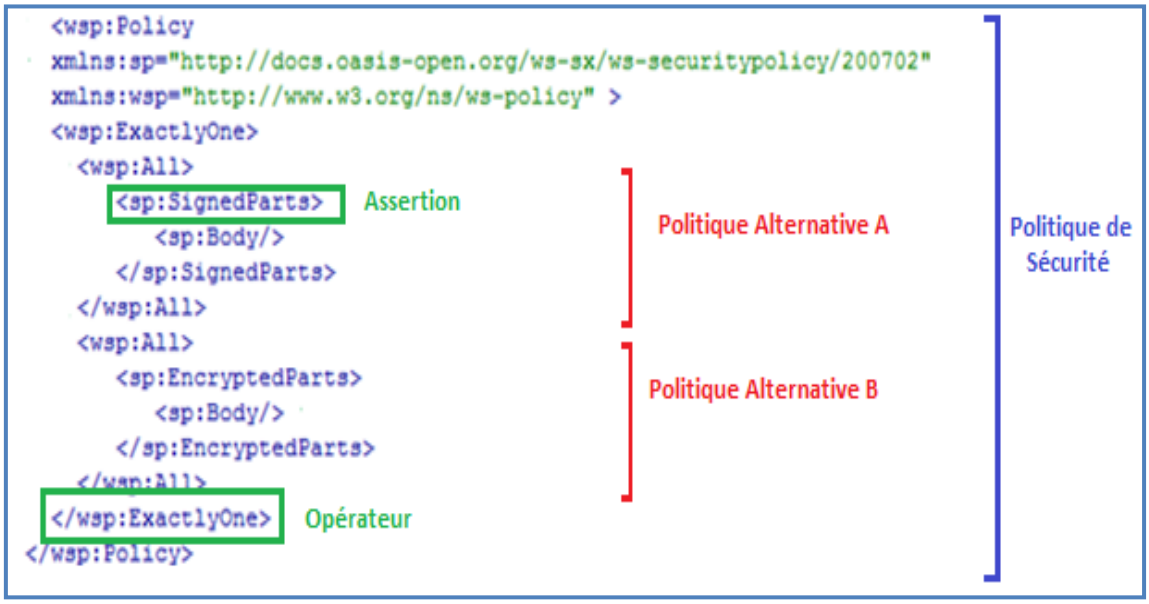


**Figure 4.11.** Exemple d'une politique de sécurité exprimée en WS-SecurityPolicy

Des opérateurs (*ExactlyOne* et *All*) sont également utilisés au niveau des politiques des services Web pour exprimer des conditions sur la manière dont les politiques alternatives et leurs assertions doivent être traitées et agrégées. Notamment, l'opérateur *All* spécifie que toutes les assertions constituant une politique alternative sont ou doivent être satisfaites selon qu'il s'agit qu'une politique de service fourni ou requi. Tandis que l'opérateur *ExactlyOne* spécifie le fait que la politique proposée (ou exigée)

satisfait (ou doit satisfaire) au moins l'une des politiques alternatives dont elle est constituée.

Pour déterminer des collections d'assertions qui couvrent chacune les aspects d'un domaine spécifique, plusieurs groupes de travaux du consortium W3C ont défini des spécifications fondées sur WS-Policy telles que WS-SecurityPolicy, WS-Reliability, etc. La figure 4.11 illustre l'utilisation de WS-Policy à travers un exemple de politique de sécurité qui intègre des assertions définies par la spécification WS-SecurityPolicy. Cet exemple décrit une politique de sécurité qui consiste à ce que le corps du message échangé par le service Web soit protégé au moins par un chiffrement ou une signature numérique. Le contenu de la balise *ExactlyOne* indique que la politique exprimée est constituée de deux politiques alternatives A et B (Figure 4.11). Chacun des contenus des balises *All* indique les assertions relatives à l'une de ces deux politiques alternatives. Chaque politique alternative ne comprend en fait qu'une seule assertion :

- ✓ l'assertion *EncryptedParts* indique que la partie du message, corps (*Body*), est ou doit bénéficier d'une protection de confidentialité (chiffrement) ;
- ✓ l'assertion *SignedParts* indique que le corps du message est ou doit bénéficier d'une protection de d'intégrité (signature numérique).

### 4.5.2. Modèle de description des politiques de service Web

Conformément à la spécification de WS-Policy 1.5, nous avons établi un modèle de politiques de services Web. Les éléments de ce modèle sont illustrés dans la figure 4.12. Chaque balise du WS-Policy est représentée dans notre modèle par une classe UML. Nous modélisons également la classe *Policy* comme une agrégation de plusieurs politiques alternatives. Une politique alternative est représentée par la classe *PolicyAlternative* et est perçue comme une agrégation d'assertions. L'assertion est modélisée par la classe *PolicyAssertion* et est décrite par un schéma XSD (*XSDType*). L'assertion exprime une propriété non-fonctionnelle représentée par la classe *NFProperty*. Chaque propriété appartient à un domaine non-fonctionnel bien déterminé *NonFunctionalDomain*. L'assertion peut être optionnelle si l'attribut booléen *optional* est mis à vrai pour exprimer le fait que la satisfaction de cette assertion n'est pas exigée. Par ailleurs, la politique porte sur une collection de sujets. Chaque sujet (*PolicySubject*)

constitue un élément du modèle concret de WSDL 2.0. Aussi, la classe *Policy* est associée à la classe *PolicyScope* qui est associée par une relation d'agrégation à la classe *PolicySubject*.

Pour satisfaire une politique *Policy*, au moins une des politiques alternatives *PolicyAlternative* contenues dans la politique doit être satisfaite. Par ailleurs, une politique alternative *PolicyAlternative* n'est considérée comme satisfaite que si toutes les assertions *PolicyAssertion* qu'elle contient et qui ne sont pas optionnelles sont satisfaites.

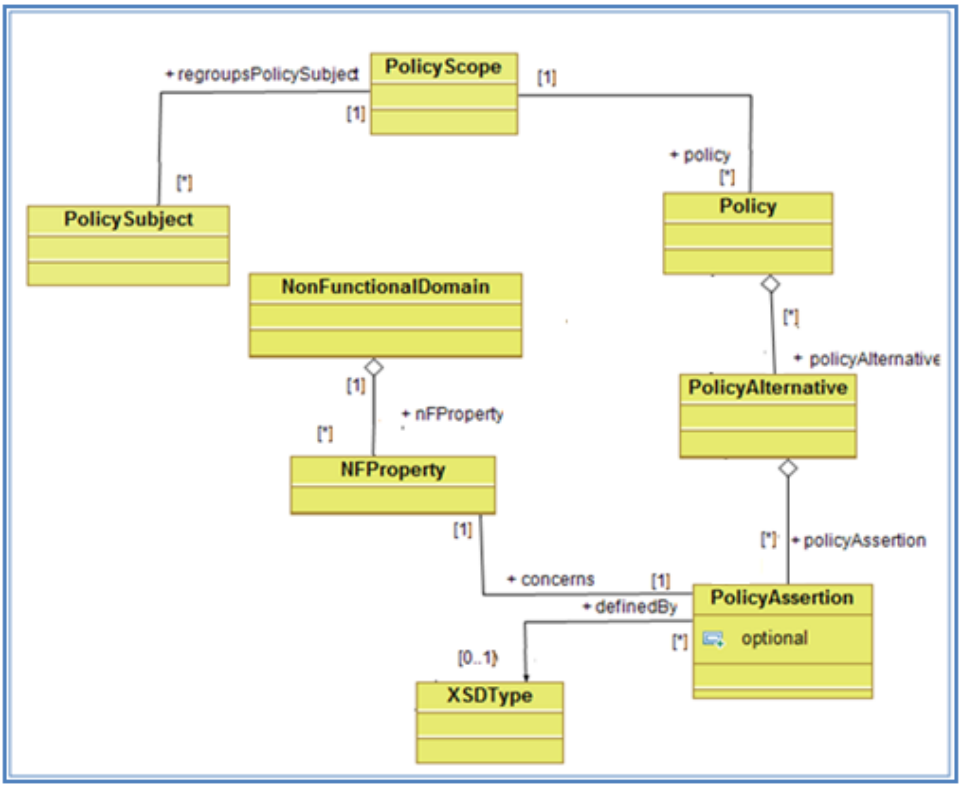


**Figure 4.12.** Modèle des politiques de services Web

### 4.5.3. Limites de WS-Policy 1.5

Les propriétés non-fonctionnelles des services Web sont très variées et dépendent du contexte de chaque fournisseur de services. Jusqu'à présent, il n'existe pas dans la littérature une liste exhaustive de ces propriétés. Néanmoins, plusieurs recherches dans le domaine de services Web [(Araujo et al., 2002), (Medina et al., 2004), (O'Sullivan et al., 2005)] ont dressé des listes de propriétés non fonctionnelles relatives à un service donné. Le travail le plus notable dans ce sens est celui mené par [O'Sullivan et al., 2005]

qui définit un ensemble de propriétés non fonctionnelles pertinentes (*Service Provider, Temporel Model, Trust, Price, Security, etc.*) et utilise les principes de modélisation ORM (*Object Role Modelling)* pour décrire leurs modèles. D'autre part, les spécifications de WS-Policy existantes, qui déterminent les assertions relatives à des domaines spécifiques, ne couvrent que quelques domaines non-fonctionnels. La qualité de service par exemple n'est pas couverte par les WS-* Spécifications.

Faute de spécifications couvrant tous les domaines non-fonctionnels et vu que les propriétés non-fonctionnelles ne peuvent être prévues de façon exhaustive, les fournisseurs et les clients de services Web sont amenés à définir leurs propres WS-Policy spécifications pour représenter leur domaines non-fonctionnels spécifiques. L'absence d'un formalise commun de définition des assertions et d'un mécanisme d'annotation de propriétés non-fonctionnelles demeurent deux limites considérables quant à l'utilisation de ce standard.

Si nous examinons les deux politiques illustrées dans la figure 4.13, nous constaterons qu'elles sont identiques. Toutefois, une comparaison automatique de ces deux politiques, et plus précisément leurs deux assertions, par un appariement syntaxique (fondé sur une comparaison des chaînes de caractères), conduit à conclure qu'elles ne sont pas équivalentes. Des mécanismes d'annotation sémantique et contextuelle restent nécessaires pour permettre de les interpréter correctement.

```
<wsp:ExactlyOne>
          <wsp:All>
          <Response Time greater than 10 ms />
          </wsp:All>
 <wsp:ExactlyOne>
```

```
<wsp:ExactlyOne>
         <wsp:All>
          <Temps de reponse superieur à 0.01 s />
         </wsp:All>
<wsp:ExactlyOne>
```

**Figure 4.13.** Exemple de deux politiques de sens identiques

### 4.5.4. Mécanisme d'annotation sémantique et contextuelle d'une assertion

Afin de permettre d'interpréter correctement des propriétés non-fonctionnelles exprimées par des assertions de WS-Policy, nous proposons d'utiliser le mécanisme d'annotation de SAWSDL pour les annoter sémantiquement (Omrana et al., 2013a). Comme précisé précédemment, l'élément ModelReference de ce standard ne peut être associé qu'aux éléments de l'interface WSDL et non à ceux du *Binding*, *Endpoint* ou *Service*. Aussi, une politique ne peut-elle être décrite directement par l'élément ModelReference de SAWSDL pour être annotée sémantiquement. De ce fait, nous exploitons la possibilité d'annoter des schémas XML des *Types* figurant dans un fichier WSDL. Il serait ainsi possible d'annoter, conformément à SAWSDL, des schémas XML de définition des assertions en les déclarant au niveau de la balise *Type*s d'un fichier WSDL.

Reconsidérons l'exemple d'assertion de WS-SecurityPolicy *SignedParts* de la figure 4.11. Il serait possible d'annoter l'élément XSD « *SignedParts* » de l'assertion par un concept sémantique d'URI http://emi/NfpOntology/#SignedMessag afin de pouvoir interpréter automatiquement le contenu de l'assertion.

```
<xs:element name="SignedParts" type="tns:SePartsType"
sawsdl:modelReference="http://emi/NfpOntology/#SignedMessag" >
<xs:annotation>
<xs:documentation xml:lang="en">5.1.1 SignedParts Assertion</xs:documentation>
</xs:annotation>
</xs:element>
```

**Figure 4.14.** Exemple d'annotation sémantique du schéma XSD de l'assertion *SignedParts*

Par ailleurs, la couverture du contexte lors de la description des propriétés non-fonctionnelles assure l'interprétation exacte des valeurs de ces propriétés. A l'instar des propriétés fonctionnelles, une propriété non-fonctionnelle peut également avoir un contexte bien défini qui permet d'évaluer correctement sa valeur. Ce contexte renseigne sur des informations telles que l'unité de mesure ou l'échelle.

Ces informations doivent être identifiées et introduites dans les descriptions des propriétés non-fonctionnelles. Selon la même logique suivie pour décrire les propriétés

contextuelles fonctionnelles, nous proposons d'intégrer ces informations au niveau des annotations sémantiques des propriétés non-fonctionnelles. Ce mécanisme peut être adopté également pour annoter les schémas XML des spécifications WS-Policy existantes.

Dans le but de permettre de décrire d'une part, la sémantique des assertions traduisant des propriétés non-fonctionnelles des services Web, et d'autre part, leurs propriétés contextuelles, nous avons proposé d'exploiter le mécanisme d'annotation de SAWSDL. Aussi, notre modèle de description des propriétés non-fonctionnelles des services Web est-il une extension du modèle de base de description des politiques issu de WS-Policy (Figure 4.15).

Notre modèle associe la classe *XSDType* aux classes *Concept* et *Instance*. Notamment, chaque schéma XSD définissant une assertion devra être annoté par un ou plusieurs concepts et instances ontologiques qui servent respectivement à fournir l'information nécessaire sur la sémantique de l'assertion et préciser son contexte.

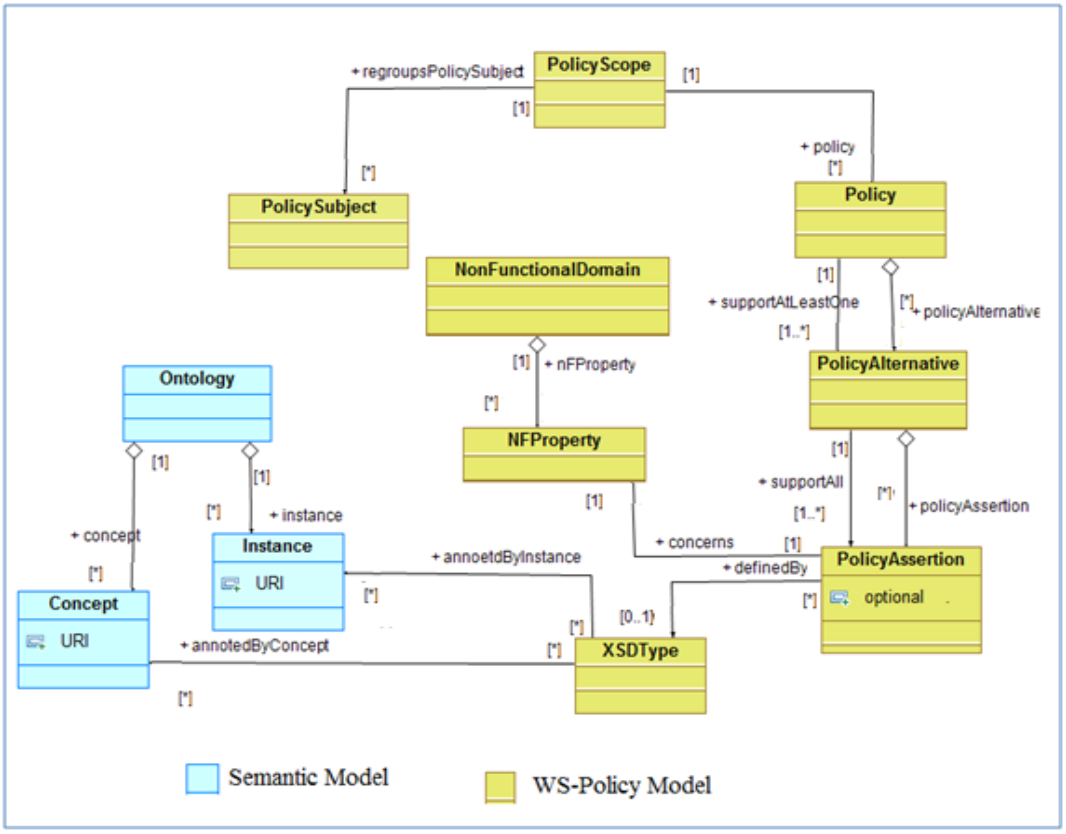


**Figure 4.15.** Notre modèle étendu de description des propriétés non-fonctionnelles

## 4.6. Synthèse

Notre modèle UML de description de services Web, entièrement aligné avec les trois standards W3C, à savoir WSDL 2.0, SAWSDL et WS-Policy 1.5, couvre les propriétés fonctionnelles, non-fonctionnelles, orientés données, contextuelles et techniques des services Web, aux niveaux syntaxique et sémantique.

Concernant l'aspect fonctionnel, nous avons exploité les mécanismes d'annotation sémantique offerts par SAWSDL afin d'étendre la description des propriétés fonctionnelles proposées par les standards WSDL 2.0 et SAWSDL, à savoir, la catégorie métier, le but fonctionnel et les entrées et sorties, aux propriétés contextuelles des entrées/sorties des opérations, ainsi qu'aux conditions associées à chaque opération. En fait, outre l'annotation sémantique associée au *Type* de l'entrée ou de la sortie d'une opération qui décrit sémantiquement le paramètre en question, nous avons proposé d'annoter ce dernier par un ensemble d'instances ontologiques qui renseignent sur son contexte fonctionnel.

Par ailleurs, l'exécution d'une opération peut être conditionnée par des contraintes liées à son exécution. En plus du concept sémantique décrivant son but fonctionnel, nous annotons également chaque opération d'un service Web par des instances sémantiques renseignant sur l'ensemble des conditions qui définissent le contexte de son exécution.

Dans le but de remédier aux limites de WS-Policy 1.5 en matière de description sémantique des politiques, nous avons proposé d'exploiter le standard SAWSDL pour annoter les schémas XSD des assertions, et de permettre ainsi d'interpréter sémantiquement et contextuellement les propriétés non-fonctionnelles définies par lesdites assertions. Notamment, nous annotons chaque schéma XSD définissant une assertion par un ou plusieurs concepts et instances ontologiques qui servent respectivement à fournir l'information nécessaire sur la sémantique de l'assertion et précisent son contexte.

Cette description riche et détaillée des services Web ouvre un large éventail de possibilités pour la découverte de ces services, leur sélection ou leur composition, et nous semble un élément de réponse au problème de l'offre pléthorique des services publiés.

Dans nos recherches, nous avons exploité ce modèle de description enrichi pour une meilleure évaluation de la composabilité offline, objet du chapitre suivant.

# Chapitre 5

# Approche de composabilité offline des services Web

## Sommaire


***5.1. Introduction ..... 117***

***5.2 Composabilité des services Web : Positionnement et enjeux ..... 118***

5.2.1 Composabilité Vs Composition ..... 119

5.2.2 Limites de l'approche classique de composabilité ..... 119

***5.3. Fondements de notre approche de composabilité des services Web ..... 120***

5.3.1 Champs et cas de composabilité ..... 120

5.3.2 Composabilité offline et online ..... 123

5.3.3 Synthèse ..... 125

***5.4. Concepts et mécanismes de la composabilité offline des services Web ..... 126***

5.4.1 Hypothèses de bases ..... 126

5.4.2 Terminologie et définitions ..... 127

5.4.4 Synthèse ..... 134

***5.5. Règles de composabilité offline ..... 135***

5.5.1 Règles de composabilité « stricte » et « souple » ..... 135

5.5.2 Règle de composabilité fonctionnelle ..... 136

5.5.3 Règle de composabilité non-fonctionnelle ..... 139

5.5.4 Règles de composabilité contextuelle ..... 142

5.5.5 Règle de composabilité orientée données ..... 144

5.5.6 Règle de composabilité technique ..... 146

***5.6 Synthèse et discussion ..... 147***

## 5.1. Introduction

Vérifier la composabilité des services Web demeure un processus peu investi par les approches de composition de ces services existant dans la littérature [(Medjahed et al., 2005), (Ernst et al., 2006), (Azmeh et al., 2011)]. La plupart des travaux existants considèrent l'appariement syntaxique ou sémantique des paramètres des services Web comme le critère de base pour leur composition. Lors de la construction du plan de composition, ils procèdent souvent à l'interconnexion de deux ou plusieurs services (ou opérations de services) suite à un appariement syntaxique et/ou sémantique de leurs entrées et sorties.

Or, en réalité ce critère demeure insuffisant pour produire des plans de composition efficaces qui peuvent donner lieu à des plans exécutables. Notamment, l'exécution de services Web indépendamment du modèle utilisé pour leur description (standards W3C, OWL-S ou WSMO, etc.), reste fortement dépendante du fichier WSDL de chaque service. A titre d'exemple, deux services utilisant des protocoles de transmission différents ne peuvent être connectés directement sans avoir recours à une solution de médiation.

A cet effet, la connexion de services ou opérations de services doit nécessairement être fondée sur un processus de vérification de la composabilité de services Web candidats qui prenne en considération des critères autres que le critère d'appariement de leurs paramètres. Ce processus doit identifier un ensemble de propriétés des services et des opérations concernés, tels que leurs paramètres d'entrées et sorties, les contraintes associées, mais aussi les détails techniques, structurels et non-fonctionnels encapsulés dans les fichiers WSDL des services Web, afin de déterminer s'il est possible de les interconnecter. En d'autres termes, il doit vérifier si deux ou plusieurs services Web sont composables à tous les niveaux (aspects ou propriétés) dont la compatibilité est strictement nécessaire pour que ces services puissent concrètement interagir les uns avec les autres. Ce niveau de détails doit être traité de manière automatique et transparente pour l'utilisateur du service composite et nécessite, en premier lieu, une description enrichie des propriétés de services, traitant à la fois les niveaux syntaxique et sémantique.

Vérifier la composabilité des services Web paraît la clé de voûte pour une composition efficace, concrète et réelle de ces services. Il constitue un défi majeur lors de la composition dynamique des services Web [Medjahed et al., 2005]. Pour permettre d'avoir des plans de composition plus efficaces, nous pensons que le processus de composition doit être fondé sur un processus de composabilité fiable qui puise ses résultats relativement à différents aspects descriptifs des services Web.

Dans le chapitre précédent, nous avons proposé une approche de description exhaustive des services Web fondés sur les standards WSDL 2.0, SAWSDL et WS-Policy 1.5 couvrant ainsi, la description fonctionnelle, non-fonctionnelle, orientée données, contextuelle et technique des services en tenant compte des aspects syntaxique et sémantique.

Sur cette base, nous proposons une approche de composabilité offline pour vérifier la possibilité de connecter deux ou plusieurs opérations de services.

Dans ce chapitre, nous introduisons d'abord, la notion de composabilité des services Web et les limites des approches existantes. Ensuite, nous présentons les fondements de notre approche tout en identifiant les types d'informations pertinentes pour la composabilité offline, objet de notre recherche. Ces informations permettent à la fois de vérifier la composabilité fonctionnelle, non-fonctionnelle, contextuelle, orientée données, et technique aux niveaux syntaxique et sémantique. Un modèle de composabilité est élaboré dans ce sens. Puis, nous détaillons l'ensemble de règles que nous avons conçues pour vérifier la composabilité de services Web [(Belouadha et al., 2010a), (Omrana et al., 2012c)].

## 5.2. Composabilité des services Web : Positionnement et enjeux

La composabilité est un aspect qui demeure peu investi dans la littérature. Dans cette section, nous commençons par cerner ce que concept tout en le positionnant par rapport au processus global de composition. Nous abordons ensuite les limites des approches de composabilité adoptées dans la littérature.

### 5.2.1 Composabilité Vs Composition

La composition des services Web (cf. Chapitre 3) est un processus complexe qui englobe plusieurs phases : découverte de services Web à composer, constitution des plans de composition, sélection du meilleur plan, etc. En pratique, la composition des services Web est une agrégation des opérations offertes par ces services et satisfaisant un besoin donné. La composabilité de services Web, quant à elle, se réfère au processus qui vérifie si un ou plusieurs services peuvent être composés. Elle constitue une étape fondamentale de la phase de constitution des plans de composition des services Web. Ce processus permet de s'assurer qu'un ou plusieurs services, voire opérations de services, peuvent être connectés pour interagir l'une avec l'autre et coopérer pour répondre au besoin fonctionnel d'une requête donnée. Un service Web *S1* peut être connecté avec un service Web *S2*, s'il existe une ou plusieurs opérations de *S1* qui peuvent être connectées avec une ou plusieurs opérations de *S2*. De ce fait, la composabilité des opérations de services Web peut être considérée comme une phase clé dont dépend l'efficacité des plans de composition élaborés.

### 5.2.2 Limites de l'approche classique de composabilité

Dans des travaux de composition étudiés [(Fujii et al., 2008), (Sohrabi et al., 2009a), (Li et al., 2011)], la composabilité des services Web se limite souvent au processus d'appariement sémantique ou syntaxique des entrées et sorties des services ou opérations concernés. Elle ne traite pas l'ensemble des informations techniques, non-fonctionnelles, contextuelles ou structurelles incorporées dans un fichier WSDL. Cependant, un processus fiable de composabilité de services doit vérifier la connectivité (ou possibilité de connexion) des opérations de services en tenant compte de l'ensemble de leurs propriétés. Notamment, l'incohérence au niveau de l'une de ces propriétés peut rendre cette connexion impossible, ou du moins, ne répondant pas de façon efficace à son objectif. En pratique, deux opérations de services Web ne peuvent être connectées que s'il existe au moins une sortie de l'une des deux opérations qui peut servir d'entrée pour l'autre opération, et que les aspects (propriétés) des deux opérations, telles que les contraintes fonctionnelles, non-fonctionnelles, techniques, etc., soient cohérents.

## 5.3. Fondements de notre approche de composabilité des services Web

Notre approche de composabilité des services Web que nous proposons est également fondée sur la composabilité des paramètres et elle distingue quatre cas en fonction du nombre de paramètres composables. Toutefois, elle intègre à cet aspect fonctionnel d'autres aspects, et étend ainsi le champ de propriétés qui peuvent former des critères de composabilité. Cette approche considère également deux types de composabilité offline et online, dans le but d'accélérer le processus global de composition des services Web. Dans la suite, nous présentons les champs et les cas aussi bien que les types de composabilité considérés dans notre approche, ce qui constituent, les fondements du processus.

### 5.3.1 Champs et cas de composabilité

Dans la littérature, la vérification de composabilité des services Web est fondée essentiellement sur le critère d'interconnexion des paramètres d'entrées/sorties de leurs opérations. Cependant, ce critère constitue un critère fondamental dont dépend la suite du processus de composabilité qui doit encore vérifier la cohérence d'autres aspects (non-fonctionnels, techniques, etc.) des opérations concernées. La cohérence de ces aspects n'est pas du tout un critère secondaire ou sans importance pour la décision de composition des services en question. Elle constitue un critère crucial dont dépend la décision ou non de composition de ces services, sauf qu'il ne convient de le vérifier qu'une fois que les opérations à interconnecter ont été identifiées à travers leurs paramètres. Aussi, notre approche de composabilité étend le champ de vérification de composabilité, limité à la vérification de connectivité de paramètres des opérations de services Web, à la vérification d'autres propriétés nécessaires pour décider de leur composition ou non, à savoir, les propriétés non-fonctionnelles, contextuelles, orientées données et techniques.

Par ailleurs, cette approche considère quatre cas de composabilité fondés essentiellement sur le degré d'interconnexion des opérations des services Web à travers leurs paramètres d'entrées/sorties. Pour désigner ces cas, nous proposons les termes suivants : composabilité totale, composabilité totale en entrée, composabilité totale en

sortie et composabilité partielle. Dans la suite, nous définissons ces termes en considérant les données i, j, n et m tels que $1 \leq i \leq n$, $1 \leq j \leq m$, et n et m sont respectivement le nombre d'entrées d'une opération A et le nombre de sorties d'une opération B.

- **Composabilité totale :** l'opération A est dite composable totalement avec B si chaque entrée $E_i$ de A peut être connectée à une sortie $S_j$ de B, et chaque sortie $S_j$ de B peut être connectée à une entrée $E_i$ de A (Figure 5.1). Dans ce cas n égale m.

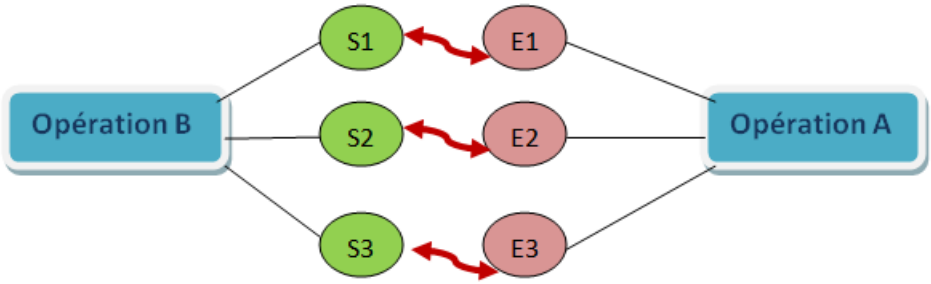


**Figure 5.2**. Composabilité totale

- **Composabilité totale en entrée :** l'opération A est dite composable totalement en entrée avec B si chaque entrée $E_i$ de A peut être connectée à une sortie $S_j$ de B, et s'il existe au moins une sortie $S_j$ de B qui ne peut être connectée à aucune entrée de A (Figure 5.2).

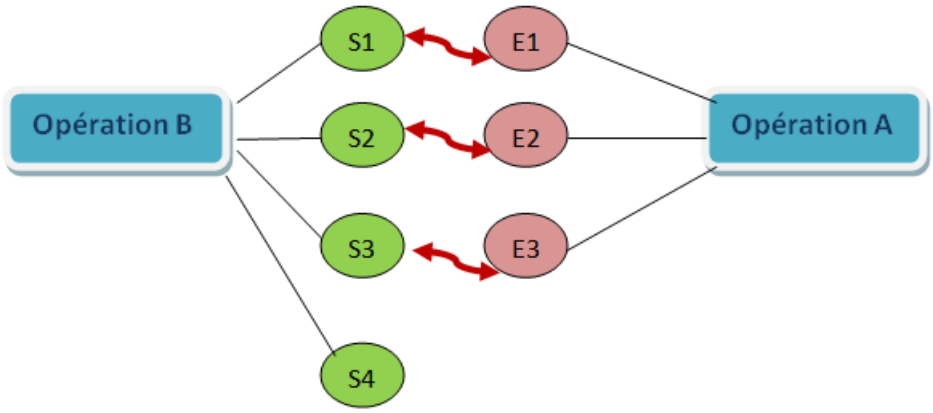


**Figure 5.3.** Composabilité totale en entrée

- **Composabilité totale en sortie :** l'opération A est dite composable totalement en sortie avec B si chaque sortie $S_j$ de B peut être connectée à une entrée $E_i$ de A, et s'il existe au moins une entrée de A qui ne peut être connectée à aucune sortie de B (Figure 5.3).

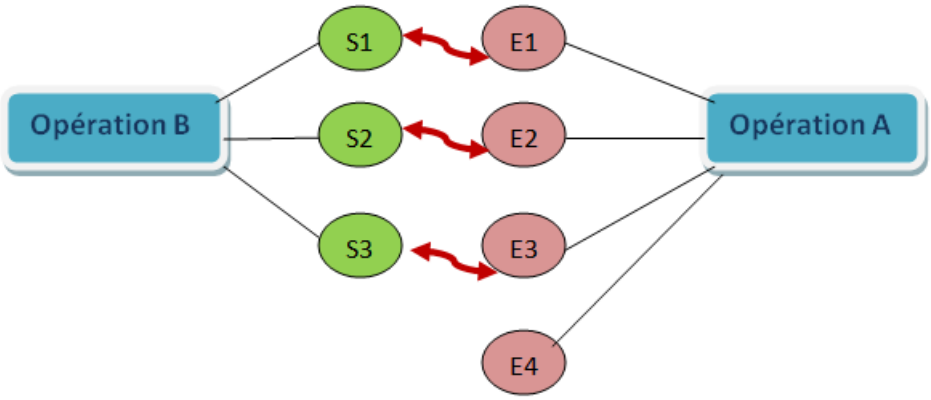


**Figure 5.4.** Composabilité totale en sortie

- **Composabilité partielle :** l'opération A est dite composable partiellement avec B si un sous ensemble non vide des entrées de A peut être connecté à un sous ensemble non vide des sorties de B, et s'il existe au moins une entrée $E_i$ de A qui ne peut être connectée à aucune sortie de B et au moins une sortie $S_j$ de B qui ne peut être connectée à aucune entrée de A (Figure 5.4).

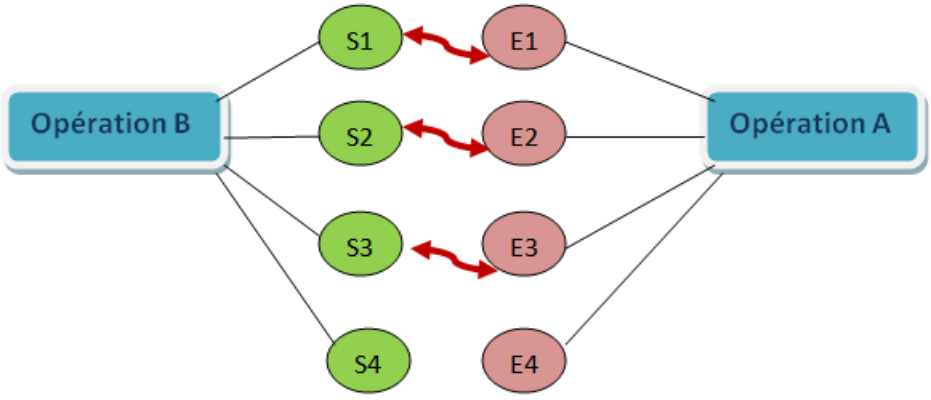


**Figure 5.4.** Composabilité partielle

Avant d'expliciter les conditions de connexion (composabilité) des paramètres de sorties et d'entrées des opérations (voir section 5.5), nous notons que nous considérons

que l'opération A est composable avec B si A est composable totalement, totalement en entrée, totalement en sortie ou partiellement avec B. Aussi, l'opération A est composable avec B ne veut pas dire systématiquement que B est composable avec A.

A l'encontre de la composabilité totale, dans les cas d'une composabilité partielle, totale en entrée ou totale en sortie, le processus de construction de plans de composition est amené à chercher de nouvelles opérations dont les paramètres peuvent être connectés aux entrées et sorties qui n'ont pu être connectées à d'autres. L'importance de considérer les quatre cas de composabilité présentés dans ce paragraphe est justifiée puisqu'ils permettent de découvrir des solutions alternatives au cas où aucune solution issue de la composabilité totale n'est trouvée pour construire des plans de composition. Or, la possibilité d'avoir plusieurs solutions alternatives à la fois fait que l'identification des cas de composabilité d'une opération en cours avec chacune des opérations alternatives reste déterminante dans le choix de l'opération à connecter à celle-ci.

Par exemple, dans le cas où l'opération A est composable totalement avec B et partiellement avec C, il sera naturellement optimal de connecter A avec B. Par ailleurs, dans le cas où A est composable totalement en entrée avec D et totalement en sortie avec E et que le processus de construction des plans de composition est fondé sur un chaînage arrière, il sera optimal de connecter A avec D (dans ce type de chaînage, le processus de composition cherche successivement à partir d'une opération en cours possédant des sorties données, de trouver d'autres opérations qui peuvent lui être connectées à travers ses entrées et qui par la suite permettent d'aboutir à ces sorties).

Enfin, nous rappelons que si l'opération A, fournie par le service *S1*, est composable avec l'opération B, fournie par le service *S2*, alors *S1* est composable avec *S2*.

### 5.3.2 Composabilité offline et online

Le processus de vérification de la composabilté des opérations des services Web s'avère un processus fastidieux qui nécessite un temps de traitement important. Il rend le processus global de composition dynamique, qui s'exécute en temps réel suite à la réception d'une requête, encore plus coûteux en temps. Pour pallier ce problème, notre approche prévoit deux phases d'exécution du processus de composabilité :

- ✓ composabilité offline (avant la réception de la requête) ;
- ✓ composabilité online (après la réception de la requête) .

L'exécution préalable du processus de composabilité a sa valeur ajoutée, en terme d'impact sur le temps de réponse du processus global de composition.

En général, la composition dynamique des services Web est initiée par une requête du client qui définit les types d'entrées renseignées et spécifie les types de sorties attendues. Nous définissons la composabilité offline comme une phase du processus de composabilité qui s'exécute au préalable de la composition dynamique des services et reste indépendante de la requête client. Dans l'objectif de contribuer à minimiser le temps de réponse du processus global de composition dynamique des services, cette phase permet d'identifier les opérations de services qu'il est possible de composer et dont les aspects, fonctionnels, non-fonctionnels, techniques, orientés données et contextuels n'empêchent pas l'exécution lorsqu'elles sont composées. Elle vérifie, indépendamment des requêtes clients, si deux opérations peuvent être connectées pour produire un service composite considéré exécutable si l'on se réfère à leurs descriptions.

La composabilité online, quant à elle, est une phase du processus de composabilité qui s'exécute au moment de la composition dynamique des services. Son objectif se limite à vérifier si deux opérations, considérées comme composables lors du processus de composabilité offline, peuvent toujours être connectées pour produire un service composite exécutable dans le contexte de la requête renseignée. Aussi, devrait-elle tenir compte d'éléments exprimés dans la requête, telles que les instances des paramètres renseignés.

Le processus de composabilité ne peut être effectué dans sa globalité avant la réception de la requête client. Notamment, le résultat final de composabilité, et par la suite de composition des services Web, dépend étroitement de la requête. Comme nous l'avons noté dans la section 5.2, l'objectif de la composabilité est de permettre au processus de composition dynamique de produire des plans de composition qui répondent à un besoin fonctionnel donné, mais qui sont exécutables dans le contexte de la requête renseignée. Considérons le cas d'un service Web composite de paiement électronique, composé de l'interconnexion de deux opérations de services Web Op1 et Op2 :

- ✓ Op1 convertit un numéro de carte bancaire en numéro de compte
- ✓ Op2 effectue le paiement à partir d'un numéro de compte.

Si Op2 de paiement a une pré-condition d'exécution qui exige que le numéro de compte soit créé par une banque localisée au niveau national, ce service ne sera pas exécutable dans le contexte d'une requête qui fournit un numéro de carte dont le numéro de compte correspondant ne satisfait pas cette condition. Dans le cas contraire, ce même service sera exécutable.

### 5.3.3 Synthèse

Notre approche de composabilité propose de traiter la composabilité offline des services Web en tenant compte des aspects fonctionnel, non-fonctionnel, orienté données, technique et contextuel, et en considérant les quatre cas de la composabilité, à savoir, la composabilité totale, la composabilité totale en entrée, la composabilité totale en sortie et la composabilité partielle

La composabilité offline décharge le processus de composition dynamique d'une charge énormément coûteuse en temps si elle est effectuée au moment du traitement des requêtes. Elle permet de constituer au préalable un registre d'opérations composables qui peut être consulté et réutilisé par ce processus pour traiter toute requête, puisque les conditions de composabilité appliquées portent sur des propriétés descriptives des services qui restent indépendantes des contraintes dynamiques des requêtes.

La constitution des plans de composition dynamique se verra ainsi limitée à la vérification de composabilité online et la génération des plans à partir du registre d'opérations composables online (composables dans le contexte de la requête reçue). De ce fait, la composabilité offline demeure un processus clé qui devrait contribuer à minimiser considérablement le temps de composition dynamique, et aussi permettre à un fournisseur de services de pouvoir au préalable détecter les incompatibilités existantes entre les opérations des services afin de les adapter pour obtenir des opérations composables ou en augmenter le nombre.

## 5.4. Concepts et mécanismes de la composabilité offline des services Web

Nous proposons un modèle de composabilité offline des services Web. La présente section expose les fondements de ce modèle. Elle présente les hypothèses de base qui doivent être vérifiées dans le cadre de cette recherche, définit ensuite les concepts utilisés pour établir les règles de composabilité offline, et termine par présenter le mécanisme d'appariement adopté pour vérifier ces règles.

### 5.4.1 Hypothèses de base

Notre approche s'inscrit dans cette optique et étend le champ de composabilité des services Web à différentes propriétés de ces composants logiciels pour assurer son efficacité. Or, un processus de composabilité efficace nécessite une description exhaustive et riche des services Web ciblés par ce processus. Aussi, notre approche de composabilité est-elle étroitement fondée sur l'approche de description des services Web que nous avons décrite dans le chapitre précédent. L'examen des propriétés descriptives identifiées montre que la plupart d'entre elles demeurent déterminantes lors du traitement de la composabilité offline des opérations de services. Par ailleurs, un processus de composabilité efficace doit également traiter la composabilité à un niveau sémantique. Cela suppose l'appariement des concepts sémantiques qui décrivent les propriétés des services. Or, cet appariement ne peut être simple que si ces concepts sémantiques émanent d'une même ontologie (un même modèle sémantique). Notamment, vérifier la composabilité des services Web dont la sémantique est décrite par différentes ontologies, comme le permet le mécanisme d'annotation du SAWSDL, reste possible mais requiert l'utilisation des techniques d'alignement d'ontologies considérées hors du périmètre de la présente recherche.

Pour l'ensemble de ces raisons, l'utilisation du formalisme de composabilité que nous proposons suppose la garantie de trois hypothèses qui sont les suivantes :

- L'ensemble des propriétés descriptives détaillées par notre approche de description des services Web, présentée dans le chapitre précédent, sont renseignées.

- Les annotations sémantiques des propriétés des services Web à composer réfèrent à une seule ontologie.
- Les schémas XSD des éléments définissant les services Web cibles sont annotés par un seul concept sémantique et par une ou plusieurs instances ontologiques qui informent sur leurs contextes, comme recommandé par notre approche de description des services.

### 5.4.2 Terminologie et définitions

La composition des services Web est en pratique une agrégation des opérations offertes par ces services et satisfaisant un besoin donné. Ce sont en fait ces opérations qui sont composées pour donner lieu à un nouveau service composite, sans que toutes les opérations des services concernés ne soient nécessairement impliquées dans la composition de ce nouveau service. De ce fait, l'opération fournie par un service Web se présente comme l'entité principale ciblée par un plan de composition. Par ailleurs, la composabilité des opérations des services Web dépend de la cohérence de leurs propriétés qui s'avèrent déterminantes pour leur composition et qui sont d'ordre fonctionnel, non-fonctionnel, orienté données, contextuel et technique.

Cette section met en avant les notations qui forment la terminologie de notre formalisme de composabilité offline et que nous utilisons par la suite pour définir les règles y afférant. La définition de chacun de ces termes est donnée ci-après.

**Définition 1 Assertion $Asser_{ikmpq}$**

L'assertion *q*, appartenant à une politique alternative *p* attachée à une politique *m*, associée à un endpoint *k* lié à une opération *i*, notée $Asser_{ikmpq}$, est définie par l'uplet (*assertSemConceptUri*, *assertContextUriList*, *optional*) où :

- *assertSemConceptUri* réfère à l'URI du concept sémantique qui décrit l'assertion $Asser_{ikmpq}$.
- *assertContextUriList* réfère aux URIs des instances qui fixent le contexte de l'assertion $Asser_{ikmpq}$.
- *optional* renseigne sur l'exigence ou non de la satisfaction de l'assertion. Si la valeur de cet attribut est « false » alors l'assertion est exigée, si elle est « true » alors

l'assertion reste optionnelle et sa satisfaction n'est pas exigée (cf. section 4.5.2 du Chapitre 4).

Chaque élément $\mathcal{eft}$ de cet uplet peut être désigné par la notation $Asser_{ikmpq}.\ \mathcal{eft}$.

**Définition 2 Politique $Policy_{ikm}$**

La politique *m* associée à un endpoint *k* lié à une opération *i*, notée $Policy_{ikm}$, est toute politique attachée indifféremment, à l'endpoint *k*, au binding associé à l'endpoint *k*, à l'opération *i* ou au service associé audit endpoint. Elle est définie par l'uplet (*id*, *policyLevel*, *PolicyAl*) où :

- *id* indique l'identificateur de la politique.
- *policyLevel* indique le niveau de la politique. Sa valeur appartient à l'énumération constituée des éléments *Endpoint*, *Binding*, *Service* et *Operation*.
- *PolicyAl* référence la liste des politiques alternatives *p* liées à la politique *m* associée à un endpoint *k* lié à une opération *i*, notées $PolicyAl_{ikmp}$ . Chaque politique alternative est une agrégation d'assertions $Asser_{ikmpq}$ où *q* est un indice d'assertion inférieur au nombre d'assertions composant la politique alternative *p*.

1. $PolicyAl_{ikm} = \{\ PolicyAl_{ikmp} /\ \ 1 \leq p \leq |PolicyAl_{ikm}|\ \}$

Chaque élément $\mathcal{eft}$ de cet uplet peut être désigné par la notation $Policy_{ikm}.\ \mathcal{eft}$.

L'ensemble des politiques liées à un endpoint *k* associé à l'opération *i* est noté $\mathcal{P}olicy_{ik}$.

$$Policy_{ik} = \{\ Policy_{ik} /\ \ 1 \leq m \leq |\ Policy_{ik}\ |\}$$

**Définition 3 Politique de niveau Message $MessagePolicy_{ijkm}$**

La politique de niveau message *m* associée à un paramètre *j* d'une opération *i* accessible à travers un endpoint *k*, notée $MessagePolicy_{ijkm}$ est définie par l'uplet (*id*, *MessagePolicyAl*) où :

- *id* indique l'identificateur de la politique.
- *MessagePolicyAl* référence la liste des politiques alternatives *p* liées à la politique de niveau message *m* attachée à un paramètre j d'une opération *i* accessible à travers un endpoint *k*, notée $MessagePolicyAl_{ijkmp}$. Chaque politique alternative est une

agrégation d'assertions $Asser_{ikmpq}$ où $q$ est un indice d'assertion inférieur au nombre d'assertions composant la politique alternative $p$.

Chaque élément $\mathscr{eft}$ de cet uplet peut être désigné par la notation $MessagePolicy_{ijkm}.\ \mathscr{eft}$.

L'ensemble des politiques de niveau message attachées au paramètre $P_{ij}$ quand l'opération $i$ est accédée à travers l'endpoint $E_{ik}$ est noté $MessagePolicy_{ijk}$

$$MessagePolicy_{ijk} = \{\ MessagePolicy_{ijkm}\ /\ 1 \leq m \leq |\ MessagePolicy_{ijk}\ |\}$$

Il est à noter que la politique d'un message peut décrire les mécanismes de protection du message au cours des échanges, tels que l'intégrité ou la confidentialité. Ces informations demeurent fondamentales, lors de l'invocation de l'opération du service.

**Définition 4 Endpoint $E_{ik}$**

L'endpoint $k$ d'une opération $i$ (noté $E_{ik}$) est l'endpoint à travers lequel l'opération $i$ fournie par le service Web en question est accessible. Il est défini par l'uplet (*uri*, *transportProtocol*, *Policy*) où :

- *uri* indique l'URI de l'endpoint $k$ à travers lequel l'opération $i$ du service en question est accessible.
- *transportProtocol* désigne le protocole de transport défini par le binding associé à l'endpoint $k$ et utilisé pour communiquer avec l'opération $i$.
- *Policy* référence la liste des politiques liées à un endpoint $k$ associé à l'opération $i$. Comme nous l'avons expliqué dans la définition 2, ces politiques peuvent être attachées indifféremment, à l'endpoint $k$, au binding associé à l'endpoint $k$, à l'opération $i$ ou au service associé audit endpoint.

Chaque élément $\mathscr{eft}$ de cet uplet peut être désigné par la notation $E_{ik}.\ \mathscr{eft}$.

L'ensemble des endpoints d'une opération $i$ est noté $E_i$ tel que :

$$E_i = \{\ E_{iK}\ /\ 1 \leq k \leq |\ E_i\ |\}$$

**Définition 5 Paramètre $P_{ij}$**

Le paramètre *j* de l'opération *i*, noté $P_{ij}$ peut être représenté par l'uplet (*id, messageLabel, semConceptUri, xsdSchema, contextUriList, exitsLoweringSchema, existsLiftingSchema, MessagePolicy*) où :

- *id* indique l'identifiant unique du paramètre $P_{ij}$.
- *messageLabel* ∈ {"In", "Out"}. La valeur "In" indique que le paramètre $P_{ij}$ est une entrée de l'opération *i* alors que la valeur "Out" indique qu'il est une sortie de cette opération.
- *semConceptUri* réfère à l'URI du concept sémantique qui décrit le paramètre $P_{ij}$.
- *xsdSchema* réfère au schéma XSD utilisé pour représenter les données relatives au paramètre $P_{ij}$.
- *contextUriList* réfère aux URIs des instances qui fixent le contexte du paramètre $P_{ij}$.
- *exitsLoweringSchema* informe sur l'existence ou non d'un schéma de mapping qui transforme une instance du concept sémantique d'URI $semConceptUri_{ij}$ en des données respectant la structure XSD du paramètre $P_{ij}$.
- *existsLiftingSchema* informe sur l'existence ou non d'un schéma de mapping qui transforme des données respectant la structure XSD du paramètre $P_{ij}$ en une instance du concept sémantique d'URI *semConceptUri*.
- *MessagePolicy* informe sur un ensemble de paires ($E_{ik}$, $MessagePolicy_{ijk}$) qui associent à chaque endpoint $E_{ik}$ de $E_i$, l'ensemble des politiques de niveau message $MessagePolicy_{ijk}$ attachées au paramètre $P_{ij}$ quand l'opération *i* est accédée à travers l'endpoint $E_{ik}$.

Chaque élément $_{e}f_{t}$ de l'uplet définissant le paramètre $P_{ij}$ peut être désigné par la notation $P_{ij}.\ _{e}f_{t}$.

**Définition 6 Opération $Op_i$**

L'opération ***i***, notée $Op_i$, est définie par l'uplet (*key*, *params*) où :

- *key* indique une clé d'identification unique de l'opération $Op_i$.
- *params* identifie l'ensemble des paramètres d'entrée et de sortie de l'opération $Op_i$.

$$params_i = \{ P_{ij} / \ 1 \leq j \leq | params_i | \}$$

Chaque élément $\mathcal{eft}$ de cet uplet peut être désigné par la notation $Op_i.\ \mathcal{eft}$.

### 5.4.3 Mécanisme d'appariement sémantique

Exploiter l'information sémantique contenue dans les descriptions de services Web est un élément essentiel pour la vérification automatique de la composabilité de leurs opérations. Aussi, le processus de composabilité doit-il utiliser un mécanisme d'appariement sémantique. Considérons le cas des paramètres d'opérations de services Web. L'interconnexion de deux opérations fournies par des services Web est, en fait, réalisée par le biais de leurs paramètres. Or, ces paramètres sont annotés par des concepts ontologiques qui peuvent être syntaxiquement différents mais sémantiquement équivalents ou similaires. Le processus de composabilité doit, notamment, tenir compte des similitudes possibles entre ces concepts sémantiques. La sortie d'une opération ne peut, en fait, être connectée à l'entrée d'une autre que si ces deux paramètres réfèrent à deux concepts carrément ou relativement équivalents du point de vue sémantique. La présente section explicite le mécanisme d'appariement des concepts sémantiques adopté pour atteindre cet objectif.

Une ontologie permet de décrire sémantiquement les concepts d'un domaine donné ainsi que leurs différentes propriétés. Ces concepts sont reliés entre eux à travers 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], à savoir, les correspondances : « exacte », « plug-in », « subsume » et « nulle ».

**Similitude « exacte»**

La similitude entre deux concepts C1 et C2, notée *Sim (C1, C2)*, est considérée comme une correspondance « exacte » si les deux concepts C1 et C2 sont équivalents, autrement dit s'ils appartiennent à la même classe ontologique (s'ils sont annotés par le même URI dans une ontologie). Pour exprimer la correspondance « exacte » des concepts C1 et C2, 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 si l'URI utilisé pour annoter le concept C1 référence, dans l'ontologie considérée, un concept défini comme une sous-classe du concept référencé par l'URI utilisé pour annoter le concept C2. Pour exprimer la correspondance « plug-in » des concepts C1 et C2, nous adoptons la notation : C1 $\subset$ C2.

**Similitude « subsume»**

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

**Similitude « nulle»**

La similitude entre deux concepts C1 et C2 est considérée comme une correspondance « nulle » 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 « nulle » des concepts C1 et C2, nous adoptons la notation : C1 $\neq$ C2.

**Relaxation sémantique**

L'approche d'appariement sémantique fondée sur la distinction des quatre catégories de correspondances sémantiques exposées ci-dessus est une approche très utilisée dans le domaine des services Web. Notre mécanisme d'appariement sémantique adopte cette approche comme mécanisme fondamental de détection de similarités sémantiques entre les concepts annotant des services Web [Omrana et al., 2012a]. En outre, dans le cas où aucune correspondance « exacte », de type « plugin » ou « subsume » n'est détectée

entre les concepts appariés, il utilise en deuxième lieu, selon le cas traité, les mécanismes de relaxation sémantique [Hatzi et al., 2009]. Ces mécanismes permettent par exemple de connecter, en plus des paramètres annotés par des concepts équivalents ou parents du point de vue sémantique, ceux annotés par des concepts sémantiquement similaires, afin d'élargir le champ de résultats possibles lors du processus de composabilité. Par exemple, soit une ontologie où le concept *Voiture* a deux sub-concepts *Voiture familiale* et *Voiture Sport* (considérés aussi comme deux concepts « plugin » du concept *Voiture*) qui ne sont reliés par aucune des trois relations « exacte », « plugin » ou « subsume » : les mécanismes de relaxation sémantique peuvent considérer ces deux concepts, qui possèdent un parent sémantique commun, comme concepts similaires relativement à un seuil de distance sémantique prédéfini par l'utilisateur.

Le mécanisme de relaxation sémantique calcule le degré de proximité de deux concepts grâce à la mesure de la distance sémantique qui les sépare dans une même ontologie. Cette contrainte (l'usage d'une même ontologie) est compatible avec les hypothèses de notre formalisme de composabilité. Par ailleurs, pour calculer la distance sémantique entre deux concepts, le mécanisme de relaxation sémantique considère le nombre d'arêtes entre des concepts pères et fils dans une ontologie (deux concepts A et B sont liés par une arête si A est une sous-classe directe de B). Dans ce contexte, [Resnik, 1999] propose de mesurer la distance sémantique globale entre deux concepts C1 et C2, en utilisant la distance qui sépare la super-classe la plus spécialisée parmi les super-classes communes entre ces deux concepts (si elle existe), à la racine de l'ontologie, et la mettre à l'échelle par la somme des distances des deux concepts par rapport à la racine de l'ontologie. Cette super-classe commune, notée *scc (C1, C2)*, interprète l'intersection la plus spécialisée que les deux concepts ont en commun (le concept parent le plus proche aux deux concepts). La distance d'un concept C par rapport à la racine de l'ontologie, notée Depth (C), est égale à la profondeur de ce concept dans l'arbre ontologique,. La formule de calcul de la distance sémantique globale entre deux concepts C1 et C2, notée diSem(C1, C2), est :

$$diSem\ (C1, C2) = 1- [2.\ Depth\ (\ scc(C1, C2))\ /\ (Depth\ (C1) + Depth\ (C2))]$$

La valeur d'une distance sémantique globale entre deux concepts C1 et C2 est comprise entre 0 et 1. La valeur 1 interprète une divergence sémantique absolue des deux concepts appariés, tandis que la valeur 0 indique leur équivalence (C1 ≡ C2). La similitude entre deux concepts C1 et C2 est considérée comme une correspondance « similaire » si la distance sémantique entre ces deux concepts est inférieure à un seuil, noté Seuil, défini par l'utilisateur. Pour l'exprimer, nous adoptons la notation C1 ≅ C2.

Afin d'illustrer le mécanisme de calcul de distance sémantique [Resnik, 1999] que nous adoptons et qui se présente en concordance avec nos hypothèses de base, nous considérons l'exemple illustré dans la figure 5.6. Cet exemple calcule la distance sémantique globale entre deux concepts F et D d'une même ontologie de racine A.

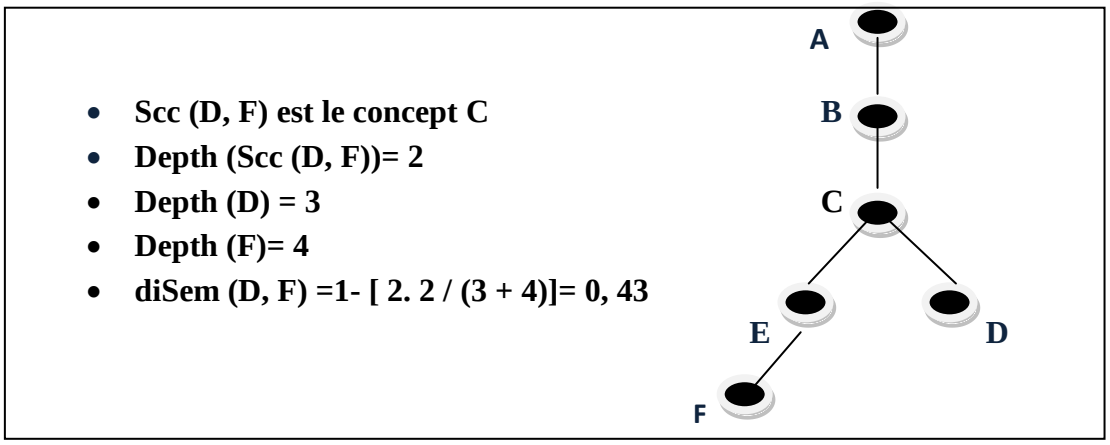


**Figure 5.5.** Exemple de calcul de distance entre deux concepts par relaxation sémantique

### 5.4.4 Synthèse

La présente section définit les concepts et mécanismes utilisés dans le processus de composabilité offline. Elle présente les hypothèses de base retenues dans le cadre de cette recherche, définit ensuite les concepts utilisés pour établir les règles de composabilité offline, et enfin expose le mécanisme d'appariement sémantique adopté pour vérifier ces règles.

## 5.5. Règles de composabilité offline

Pour interconnecter les opérations des services Web, six règles doivent être appliqué afin de vérifier leur composabilité offline. Chacune de ces règles traduit un ensemble de conditions qui seront vérifiées automatiquement par le processus de composabilité offline dans le but de s'assurer de la possibilité d'interconnexion de deux opérations données. Dans la suite, nous présentons les types de règles élaborées à cette fin, à savoir, les règles de composabilité fonctionnelle, non-fonctionnelle, orientée donnée, technique et contextuelle, tout en introduisant les notions de règles de composabilité strictes et souples [Omrana et al., 2013c].

### 5.5.1 Règles de composabilité « stricte » et « souple »

Nous proposons d'introduire les notions de règles « stricte » et « souple ». Nous considérons qu'une règle de composabilité de deux opérations peut être désignée comme règle «stricte» dans le cas où l'interconnexion de ces deux opérations par le biais de deux paramètres n'est possible que si la règle est satisfaite.

Par ailleurs, nous considérons qu'une règle de composabilité de deux opérations pouvant être interconnectées par le biais de deux paramètres, peut être désignée comme règle « souple » si l'utilisation d'une solution de médiation manuelle, semi-automatique ou automatique peut être envisageable dans le but d'aboutir à construire un plan de composition exécutable, même si la règle n'est pas satisfaite. Autrement dit, nous considérons que les règles portant sur la composabilité des paramètres du point de vue fonctionnel sont des règles « strictes » puisque toute incompatibilité fonctionnelle de deux paramètres rend l'interconnexion d'opérations par le biais de ces paramètres logiquement impossible. Par ailleurs, nous considérons qu'une règle est « souple » lorsqu'il est possible d'avoir recours à un médiateur pour apporter des adaptations au niveau des aspects incompatibles, et ce dans le but de faire interagir les opérations qu'il s'avère possible d'interconnecter à travers des paramètres. Un exemple d'aspect pouvant être sujet d'incompatibilité est le format de données des paramètres qu'il est possible de transformer pour réaliser l'adaptation nécessaire. Les solutions de médiation permettent en fait d'aboutir à des plans de composition exécutables quand aucun des plans produits ne l'est initialement. En d'autres termes, lorsque le processus de composabilité détecte qu'une règle qualifiée de « souple » n'est pas satisfaite pour deux

opérations données, il serait tout de même possible d'opter pour la composition de ces deux opérations à condition d'utiliser une solution de médiation permettant de rendre le service Web composite exécutable.

### 5.5.2 Règle de composabilité fonctionnelle

L'aspect fonctionnel d'un service spécifie ce que peut offrir ce service à ses clients en termes de fonctionnalités. Chaque opération peut être définie par un ensemble de propriétés fonctionnelles :

- ✓ la catégorie métier
- ✓ la fonctionnalité offerte par cette opération,
- ✓ ses entrées/sorties
- ✓ ses conditions

La sémantique de tous ces éléments est renseignée au niveau d'un fichier SAWSDL grâce à l'attribut *ModelReference*.

**Propriétés fonctionnelles impliquées dans la composabilité offline fonctionnelle**

La catégorie de service et le but fonctionnel sont deux éléments décisifs lorsqu'il s'agit du processus d'appariement (*matching*) des services Web, qui vise à en dégager ceux équivalents ou similaires. Toutefois, quand il est question de vérifier leur composabilité qu'elle soit offline ou online, ces deux propriétés fonctionnelles restent sans grand impact sur le résultat. Nous estimons que deux opérations peuvent être composables même si leurs catégories de services ou leurs fonctionnalités offertes ne sont pas sémantiquement équivalentes ou similaires. Notamment, pour répondre à une requête donnée, il est parfois nécessaire, de par la nature de cette requête, de composer des opérations de services Web issus de catégories métier différentes, et évidemment offrant des fonctionnalités différentes qui se complètent dans un cadre coopératif pour accomplir la fonctionnalité souhaitée. Considérons l'exemple d'une opération X de catégorie de service « Administration publique » qui offre comme fonctionnalité l'envoi de demande d'emploi pour un poste administratif et qui prend comme entrée un sommaire de l'expérience professionnelle. Cette opération peut être par exemple liée à une opération Y de catégorie de service « Education » et qui offre comme fonctionnalité la correction grammaticale d'un texte et fournit comme sortie un texte corrigé

grammaticalement, si l'objectif fonctionnel que l'on cherche à atteindre consiste à utiliser un service qui permet de corriger le texte d'une demande d'emploi et de la déposer pour un poste administratif. Les deux opérations peuvent être composées pour répondre à l'objectif visé sans qu'aucune restriction ne soit faite sur la cohérence de leurs catégories métier ou leur fonctionnalités.

Par ailleurs, la composabilité de deux opérations relativement à leurs conditions ne peut être vérifiée qu'au moment de la composabilité online, puisque la validité ou non d'une condition associée à une opération ne peut être connue que dans le contexte où elle est appliquée. Notamment, la validité ou non de cette condition dépend de la requête et ne peut être déterminée qu'après avoir les données de cette requête. Or, la composabilité de deux opérations relativement à leurs conditions requiert que ces conditions soient valides lors de leur interconnexion pour ne pas empêcher leur exécution, chose qui n'est possible de vérifier que dans le contexte de la requête.

Considérons, une opération Op1 qui effectue un paiement à partir d'un numéro de compte et qui possède comme pré-condition d'exécution le fait que ce numéro de compte soit valide (créé par une banque localisée au niveau national). Elle est composable avec une opération Op2 qui fournit, à partir d'un numéro de carte, le numéro de compte correspondant, si ce dernier s'avère un numéro de compte valide.

Op1 et Op2 ne sont pas considérées comme composables relativement à leurs conditions, s'il s'avère que le numéro de compte correspondant au numéro de carte renseigné n'est pas valide (créé par une banque localisée à l'étranger). Dans un tel cas, l'opération de paiement ne peut s'exécuter. L'interconnexion de ces deux opérations répond bien à l'objectif fonctionnel de paiement sauf que la condition associée à l'une des deux rend leur composabilité dépendante de l'instance du paramètre *numéro de carte* renseignée au niveau de la requête.

Dans ce sens, nous considérons que la composabilité de deux opérations relativement à leurs conditions ne peut être vérifiée qu'au moment de l'exécution de la requête, et est par la suite une tâche du processus de composabilité online.

**Critères retenus de composabilité offline fonctionnelle**

Suite aux constats évoqués au niveau de ce paragraphe, le processus de composabilité fonctionnelle que nous proposons ne considère que deux opérations de services Web sont composables par le biais de leurs paramètres, que lorsque l'une de deux conditions, fondées sur l'appariement de ces paramètres, est vérifiée.

- ✓ La première condition vérifie que la correspondance sémantique entre au moins le concept annotant une entrée de l'une des deux opérations et celui annotant une sortie de l'autre est de type « exacte » ou « subsume ». L'entrée en question doit être équivalente ou encapsuler la sortie concernée du point de vue sémantique. Cette condition trouve ses fondements dans le fait qu'une opération ne peut être exécutée que lorsque l'on met à sa disposition des entrées qui appartiennent à l'éventail d'entrées qu'elle attend. A titre d'exemple, une opération qui attend comme entrée une liste de véhicules admettra en entrée tout type de véhicules et peut être connectée à une opération qui fournit comme sortie une liste de voitures ou de camions. Ce qui ne convient pas (et est à éviter) dans le cas inverse : quand une opération attend comme entrée une liste de voitures, il ne convient généralement pas de la connecter à une opération qui fournit comme sortie une liste de véhicules de tout type (qui comprend des voitures aussi bien que des camions par exemple).

- ✓ La deuxième condition consiste à ce que la distance sémantique entre les concepts appariés soit inférieure à un seuil fourni par l'utilisateur lui même. Cette condition est vérifiée dans les cas où la correspondance entre le concept annotant l'entrée de l'une des deux opérations et le concept annotant la sortie de l'autre opération n'est ni « exacte » ni de type « subsume ». Dans un tel cas, notre mécanisme d'appariement sémantique procède au calcul de leur distance selon le mécanisme de relaxation sémantique et la formule de calcul présentés dans la section 5.4.

En conclusion, la règle de composabilité fonctionnelle constituée des deux conditions citées ci-dessus peut être définie comme montré dans la figure 5.6 ci-après.

Soient les deux opérations : $Op_i$ et $Op_j$ et le seuil de distance sémantique prédéfini par l'utilisateur : *Seuil*

$Op_i$ est dite composable offline fonctionnellement avec $Op_j$ si :

$\exists (P_{im}, P_{jk}) / (P_{im}$ . messageLabel="In" $\wedge$ $P_{jk}$ . messageLabel="Out")

$\wedge$ (

($P_{im}$ . semConceptUri $\equiv$ $P_{jk}$ . semConceptUri)

$\vee$ ($P_{im}$ . semConceptUri $\supset$ $P_{jk}$ . semConceptUri)

$\vee$ (diSem **(**$P_{im}$ . semConceptUri , $P_{ik}$ . semConceptUri) <= *Seuil*)

**Figure 5.6.** Règle de composabilité fonctionnelle

La règle de composabilité fonctionnelle est considérée comme une règle « stricte ». Si elle n'est pas satisfaite, l'interconnexion des deux opérations concernées ne peut être envisagée.

### 5.5.3 Règle de composabilité non-fonctionnelle

Plusieurs travaux sur la composition des services Web utilisent les propriétés non-fonctionnelles pour la sélection du meilleur plan de composition, sauf que dans certains cas, la validité ou non du plan de composition lui même dépend de ces propriétés. Il convient dans ces cas de vérifier la cohérence des propriétés non-fonctionnelles des opérations des services à composer, au moment de la vérification de leur composabilité. Un cas typique qui illustre ce constat est celui d'une opération qui a un paramètre en entrée, associé à une politique de sécurité exigeant que ce dernier soit crypté. Cette opération ne peut être connectée à travers l'entrée concernée qu'à une autre opération qui fournit une sortie sémantiquement similaire avec cette entrée, mais nécessairement cryptée. La composer avec une autre opération qui ne satisfait pas ce critère donne lieu à un plan de composition invalide.

Pour résumer, quand il est question de répondre à une requête complexe, le processus de composition de services Web cherche en principe à composer des services qui peuvent coopérer pour satisfaire en premier lieu le besoin fonctionnel spécifié par cette requête. Les propriétés non-fonctionnelles ne sont en général considérées qu'au moment de la sélection du meilleur plan de composition dans le but d'essayer de satisfaire au

maximum des contraintes non-fonctionnelles spécifiques des clients. Aussi, considérons-nous que le processus de composabilité offline ne devrait vérifier la composabilité non-fonctionnelle des opérations que dans le sens de détecter les opérations qui ne sont catégoriquement pas composables pour des raisons d'incohérence de leurs propriétés non-fonctionnelles. Cela permet d'éviter de proposer des plans de composition invalides, comprenant des composants qui sont en réalité non composables vu l'hétérogénéité de leurs contraintes non-fonctionnelles. Cependant, l'adéquation ou non des opérations considérées comme composables offline du point de vue non-fonctionnel (i.e. respect les contraintes non-fonctionnelles des clients), ne peut être vérifiée qu'au niveau de la composition online (quand la requête, et par la suite les besoins spécifiques du client sont exprimés).

**Propriétés non-fonctionnelles impliquées dans la composabilité offline non-fonctionnelles**

L'assertion est la propriété non-fonctionnelle élémentaire utilisée pour décrire l'aspect non-fonctionnel d'un service Web dans WS-Policy. La politique est une agrégation des politique alternatives et la politique alternative, à son tour, est une agrégation d'assertions.

L'architecture WSDL 2.0 permet d'associer plusieurs implémentations concrètes à une même spécification abstraite du service Web. Les politiques, dans WS-Policy, ne peuvent être attachées qu'aux éléments de la partie concrète (technique) du service, à savoir le *Binding* y compris l'ensemble des éléments (opération, entrée, sortie) qu'il encapsule, le *Service* et l'*Endpoint*. Cette logique offre l'avantage de pouvoir publier la même fonctionnalité avec différentes contraintes non-fonctionnelles. La composabilité non-fonctionnelle de deux opérations composables du point de vue fonctionnel par le biais de deux paramètres, ne peut être constatée que si, d'une part, elles sont accessibles respectivement via au moins deux endpoints avec deux politiques qui sont cohérentes (composables) à tous les niveaux : *Endpoint, Binding, Service* et *Operation*, et si d'autre part, les politiques de niveau message attachées à leurs deux paramètres de connexion sont cohérentes (composables).

**Critères retenus de composabilité offline non-fonctionnelle**

Pratiquement, nous considérons que deux politiques générales (attachées à des endpoints) de niveau message (attachées à des paramètres), concernant deux opérations

possédant respectivement deux paramètres d'entrée et de sortie à interconnecter, sont cohérentes (composables) si elles sont constituées respectivement d'au moins deux politiques alternatives qui le sont. En d'autres termes, si chacune des assertions exigées par la politique alternative correspondant à l'opération possédant le paramètre d'entrée est cohérente (composable) avec l'une des assertions proposées par la politique alternative correspondant à l'opération possédant le paramètre de sortie. Cela signifie que chaque concept sémantique, annotant l'une des assertions exigées par la politique alternative correspondant à l'opération possédant le paramètre d'entrée doit être équivalent au, « subsume », ou similaire au concept sémantique annotant l'une des assertions proposées par la politique alternative correspondant à l'opération possédant le paramètre de sortie. En plus, les deux politiques générales doivent être cohérentes (composables) à tous les niveaux : *Endpoint*, *Binding*, *Operation* et *Service*.

La règle de composabilité offline non-fonctionnelle est une règle « souple » puisque des solutions de médiation peuvent être envisagées pour permettre aux opérations fournies par des services Web d'interagir même si leurs politiques exigées sont différentes.

Ainsi, deux opérations composables fonctionnellement par le biais de deux paramètres ne sont-elles considérées (par notre processus de composabilité offline) comme composables du point de vue non-fonctionnel que si les conditions formulées dans la figure 5.7 sont vérifiées. La vérification de ces conditions est fondée sur l'appariement sémantique des assertions exigées des politiques de niveau *Message* associées aux deux paramètres à interconnecter, ainsi que l'appariement sémantique des assertions exigées des politiques (de niveaux *Endpoint*, *Binding*, *Operation* et *Service)* associées aux deux opérations à composer.

Soient les deux opérations : $Op_i$ et $Op_j$ composables offline fonctionnellement par le biais respectivement des paramètres $P_{ix}$ en entrée et $P_{jy}$ en sortie, et le seuil de distance sémantique prédéfini par l'utilisateur : *Seuil*

$Op_i$ est dite composable offline non-fonctionnellement avec $Op_j$ par le biais respectivement des paramètres $P_{ix}$ en entrée et $P_{jy}$ en sortie si :

$\exists (E_{ik}, E_{jk'}) /$

$(\forall\ Policy_{ikm}\ ,\ \exists\ Policy_{jk'm'}\ /\ (Policy_{ikm}.\ \text{policyLevel} = Policy_{jk'm'}.\ \text{policyLevel}),$

$\exists (PolicyAl_{ikml},\ PolicyAl_{jk'm'l'}) /$

$(\forall\ \text{Asser}_{ikmlq}\ /\ (Asser_{ikmlq}.\ \text{optional} = \text{'false'})\ ,\ \exists\ Asser_{jk'm'l'q'}\ /$

$(Asser_{ikmlq}.\ assertSemConceptUri \equiv Asser_{jk'm'l'q'}.\ assertSemConceptUri)\ \boldsymbol{V}$

$(Asser_{ikmlq}.\ assertSemConceptUri \supset Asser_{jk'm'l'q'}.\ assertSemConceptUri)\ \boldsymbol{V}$

$(\text{diSem}(Asser_{ikmlq}.assertSemConceptUri, Asser_{jk'm'l'q'}.assertSemConceptUri) \leq Seuil))))$

$\wedge$

$(\forall\ MessagePolicy_{ixkw}\ ,\ \exists\ MessagePolicy_{jyk'w'}\ /$

$\exists (MessagePolicyAl_{ixkwt}, MessagePolicyAl_{jyk'w't'})\ ,$

$\forall\ Asser_{ixkwtr}\ /\ (Asser_{ixkwtr}.\ \text{optional} = \text{'false'})\ ,\ \exists\ Asser_{jyk'w't'r'}\ /$

$(Asser_{ixkwtr}.\ \text{assertSemConceptUri} \equiv Asser_{jyk'w't'r'}.\ \text{assertSemConceptUri})\ V$

$(Asser_{ixkwtr}.\ \text{assertSemConceptUri} \supset Asser_{jyk'w't'r'}.\ \text{assertSemConceptUri})\ V$

$(\text{diSem}(Asser_{ixkwtr}.\text{assertSemConceptUri}, Asser_{jyk'w't'r'}.\text{assertSemConceptUri}) \leq Seuil)))$

**Figure 5.7.** Règles de composabilité non-fonctionnelle

### 5.5.4 Règles de composabilité contextuelle

Comme explicité au niveau du chapitre 4, notre approche propose de renseigner les propriétés contextuelles, servant à décrire l'aspect fonctionnel et non-fonctionnel d'un service Web sans ambiguïté. Ces propriétés sont décrites par le biais de la balise SAWSDL ModelReference et peuvent notamment être traduites par une liste

d'instances sémantiques utilisées pour annoter les paramètres des opérations ou encore des assertions exprimant des politiques.

**Composabilité offline contextuelle pour l'aspect fonctionnel**

La composabilité offline contextuelle pour l'aspect fonctionnel doit vérifier pour deux opérations $Op_i$ et $Op_j$ qui sont fonctionnellement composables à travers, respectivement, leurs paramètres d'entrée $P_{ik}$ et de sortie $P_{jm}$, que le contexte du paramètre $P_{jm}$ est compatible avec celui du paramètre $P_{ik}$. Autrement, elle doit vérifier que chaque instance parmi les instances décrivant le contexte du paramètre $P_{ik}$, existe dans la liste des instances décrivant le contexte du paramètre $P_{jm}$ (Figure 5.8).

Soient les opérations : $Op_i$ et $Op_j$ composables offline fonctionnellement par le biais respectivement des paramètres $P_{ik}$ en entrée et $P_{jm}$ en sortie.

$Op_i$ et $Op_j$ sont dites composables offline contextuellement par le biais respectivement des paramètres $P_{ik}$ en entrée et $P_{jm}$ en sortie si :
$\forall U \in P_{ik}$ . contextUriList, $U \in P_{jm}$ . contextUriList

**Figure 5.8.** Règle de composabilité contextuelle pour l'aspect fonctionnel

**Composabilité offline contextuelle pour l'aspect non-fonctionnel**

La composabilité offline contextuelle pour l'aspect non-fonctionnel, doit vérifier pour deux opérations composables du point de vue fonctionnel par le biais de deux paramètres et supposées composables aussi du point de vue non-fonctionnel, que toutes les paires d'assertions X et Y qui assurent leur composabilité non-fonctionnelle ont le même contexte (autrement, elle doit vérifier que Y satisfait X). Autrement, elle doit vérifier que chaque instance parmi les instances décrivant le contexte de l'assertion X, existe dans la liste des instances décrivant le contexte de l'assertion Y.

Soient les deux opérations : $Op_i$ et $Op_j$ composables offline fonctionnellement par le biais respectivement des paramètres $P_{ix}$ en entrée et $P_{jy}$ en sortie, et aussi non-fonctionnellement à travers respectivement les endpoints $E_{ik}$ et $E_{jk'}$ par le biais des politiques alternatives $PolicyAl_{ikml}$ et $PolicyAl_{jk'm'l'}$ et des politiques alternatives de niveau message $MessagePolicyAl_{ixkwt}$ et $MessagePolicyAl_{jyk'w't'}$

$Op_i$ et $Op_j$ sont dites composables offline contextuellement du point de vue non-fonctionnel, par le biais respectivement des paramètres $P_{ik}$ en entrée et $P_{jm}$ en sortie si :

$$(\forall Asser_{ikmlq}, \exists Asser_{jk'm'l'q'} /$$

$$\forall U \in Asser_{ikmlq}.\text{assertContextUriList}, U \in Asser_{jk'm'l'q'}.\text{assertContextUriList})$$

$$\wedge$$

$$(\forall Asser_{ixkwtq}, \exists Asser_{jyk'w't'q'} /$$

$$\forall U \in Asser_{ixkwtq}.\text{assertContextUriList}, U \in Asser_{jyk'w't'q'}.\text{assertContextUriList})$$

**Figure 5.9.** Règle de composabilité contextuelle pour l'aspect non-fonctionnel

Les deux règles de composabilité contextuelle sont considérées comme des règles « souples ». Des solutions de médiation pour l'adaptation des contextes peuvent être mises en place.

### 5.5.5 Règle de composabilité orientée données

Il ne fait nul doute que deux opérations composables fonctionnellement par le biais de leurs paramètres (annotés par des concepts équivalents, similaires sémantiquement ou dont l'appariement est de type « subsume ») peuvent échanger avec succès les données relatives à ces paramètres si leurs schémas XSD sont syntaxiquement équivalents, autrement dit si ces paramètres possèdent le même schéma XSD. Cependant, les schémas XSD de deux paramètres d'entrée X et de sortie Y, par le biais desquels deux opérations sont composables fonctionnellement, peuvent être différents en pratique.

Dans ce contexte, la composabilité orientée données a pour but de vérifier la possibilité de connecter deux paramètres de deux opérations composables fonctionnellement, en tenant compte de la conformité de leurs types de données ou en s'assurant qu'il y a moyen de faire interagir ces deux opérations à travers un échange de données compatibles. Le cas échéant, les deux opérations ne peuvent être connectées que s'il

existe une solution de médiation qui convertit les données du paramètre de sortie Y conformément au type de données de l'entrée X.

Par ailleurs, les annotations SAWSDL permettent de décrire les entrées et sorties des services Web en utilisant des concepts sémantiques, et de définir des schémas de *mapping* (*LoweringSchema* et *LiftingSchema*) assurant la transformation de données sémantiques vers un format XML et vice versa. Ces schémas de *mapping* quand ils sont définis (quand ils existent), jouent un rôle déterminant lorsqu'il est question de faire connecter deux opérations fonctionnellement composables. Grâce à ces schémas, ces opérations peuvent notamment être composables par le biais de deux paramètres d'entrée X et de sortie Y, annotés par un même concept ou des concepts équivalents dans une ontologie donnée, et décrits par deux schémas XSD différents mais en réalité conformes à un même ensemble de données sémantiques. Les schémas de *mapping* assurent, dans ce cas, le rôle du médiateur qui permet, d'une part, de transformer les données XML conformes aux XSD du paramètre Y en données sémantiques grâce aux *LiftingSchema* associé au paramètre Y, et d'autre part, de transformer ces mêmes données sémantiques en données XML conformes au XSD du paramètre X grâce aux *LoweringSchema* associé au paramètre X. Cela permet aux deux opérations de pouvoir communiquer, lorsqu'elles sont composées, en échangeant des données représentées par des schémas XSD différents.

Ce constat n'est pas valide dans le cas où les schémas des paramètres X et Y sont différents, alors que ceux-ci sont annotés par des concepts similaires sémantiquement ou dont l'appariement est de type « subsume ». Les schémas de mapping ne peuvent être considérés comme moyen pour s'assurer de la composabilité orientée données des opérations en jeu puisqu'il n'est pas évident que ces paramètres soient représentées par le même ensemble de données sémantiques.

En conclusion, comme illustré dans la figure 5.10, deux opérations Op*i* et Op*j*, composables fonctionnellement par le biais de leur deux paramètres d'entrée $P_{ik}$ et de sortie $P_{jm}$, sont dites composables offline au niveau données par le biais de ces deux paramètres, si les schémas XSD de ces derniers sont syntaxiquement équivalents, ou dans le cas où ces deux paramètres sont annotées par des concepts équivalents et les

lowering schema et lifting schema, respectivement des paramètres $P_{ik}$ et $P_{jm}$, sont disponibles.

Soient les opérations $Op_i$ et $Op_j$ composables offline fonctionnellement par le biais respectivement des paramètres $P_{ik}$ en entrée et $P_{jm}$ en sortie

$Op_i$ et $Op_j$ sont dites composables offline au niveau données par le biais respectivement des paramètres $P_{ik}$ en entrée et $P_{jm}$ en sortie si :

$$( P_{ik} . \text{xsdSchema} \equiv P_{jm} . \text{xsdSchema} )$$

$$\vee \quad ((P_{ik} .\text{semConceptUri} \equiv P_{jm} .\text{semConceptUri}) \wedge (P_{ik} .\text{existsLoweringSchema=true})$$

$$\wedge \quad (P_{jm} . \text{existsLiftingSchema} = \text{true} ) )$$

**Figure 5.10.** Règle de composabilité orientée données

La règle de composabilité orientée données est considérée comme une règle « souple ». Des solutions médiatrices de transformation de données peuvent être envisagées.

### 5.5.6 Règle de composabilité technique

L'information d'ordre technique concernant un service Web est renseignée au niveau de l'élément *Binding* de son fichier WSDL. Cet élément sert à identifier le protocole de transmission (par exemple, SOAP ou HTTP), et par la suite le format de message concret, à utiliser pour accéder au service à travers un endpoint donné. Aussi, deux services Web ne peuvent-ils communiquer qu'à travers deux endpoints qui utilisent des liaisons fondées sur le même protocole de transport. De ce fait, la règle de composabilité offline technique que nous adoptons considère, comme illustré dans la figure 5.11, que deux opérations de services Web composables du point de vue fonctionnel par le biais de deux paramètres, sont dites composables techniquement si et seulement si elles possèdent respectivement au moins deux endpoints dont les protocoles de transmission associés sont similaires.

Soient les opérations : $Op_i$ et $Op_j$ composables offline fonctionnellement par le biais de deux paramètres.

$Op_i$ est dite composable offline techniquement avec $Op_j$ si et seulement si :

$$\exists (E_{ik}, E_{jk'}) \quad 1 \leq k \leq |E_i| \text{ et } 1 \leq k' \leq |E_j| \; / \; E_{ik}.\text{transportProtocol} = E_{jk'}.\text{transportProtocol}$$

**Figure 5.11.** Règle de composabilité technique

La règle de composabilité technique est considérée comme une règle « souple ». Des solutions de médiation peuvent être envisagées pour adapter le protocole de communication selon le besoin du service Web client.

## 5.6 Synthèse et discussion

La composabilité de services Web est une étape déterminante lors du traitement de la composition des services. Dans le présent chapitre, nous avons proposé une approche de composabilité offline qui vérifie au préalable la possibilité de connecter deux ou plusieurs opérations de services sur la base des informations descriptives des services Web, recueillis selon notre modèle de description aligné avec les standards W3C. Notre approche de composabilité considère les aspects fonctionnel, non-fonctionnel, contextuel, orienté données et technique, et tient compte des niveaux syntaxique et sémantique. Dans ce sens, six règles de composabilité offline ont été définies pour chaque aspect :

- ✓ Règle de composabilité offline fonctionnelle
- ✓ Règle de composabilité offline non-fonctionnelle
- ✓ Règle de composabilité offline contextuelle pour l'aspect fonctionnel
- ✓ Règle de composabilité offline contextuelle pour l'aspect non-fonctionnel
- ✓ Règle de composabilité offline orientée données
- ✓ Règle de composabilité offline technique

Les notions de règles « stricte » et « souple » ont été proposées pour différencier entre une règle dont la non satisfaction empêche toute possibilité d'interconnecter deux opérations (le cas de la règle de composabilité offline fonctionnelle), et une règle dont la non satisfaction n'empêche pas la possibilité d'interconnecter deux opérations si une solution de médiation manuelle, semi-automatique ou automatique peut être exploitée, (les cas des règles de composabilité offline non-fonctionnelle, contextuelle, orienté données et technique). C'est pour cette raison que la vérification de la règle fonctionnelle doit précéder la vérification de toutes les autres règles.

Les informations descriptives des services Web impliquées dans notre processus de composabilité sont celles considérées pertinentes à cet effet. Des propriétés sémantiques d'ordre fonctionnel (telles que la catégorie du service utilisée par [Medjahed et al.,

2005] pour la vérification de la composabilité de deux opérations) ont été exclues par notre approche, dans la mesure où nous considérons que ces propriétés sont sans grand impact sur la composabilité des opérations. Notamment, deux opérations de catégories de services disjointes peuvent être connectées. L'exclusion d'autres propriétés fonctionnelles telles que les pré-conditions, les post-conditions ou les effets, est justifiée par le fait qu'elles ne doivent tout simplement être prises en considération que lors de la phase de composabilité online, durant laquelle les contraintes correspondant à ces aspects sont spécifiées par la requête à traiter.

Par ailleurs, le périmètre de la composabilité couvert par notre approche traite la composabilité totale, partielle, totale en entrée et totale en sortie de deux opérations, au moment où certains travaux connexes [(Medjahed et al., 2005), (Azmeh et al., 2011)] se limitent à la composabilité totale de deux opérations. Cela permet d'aboutir à des plans de composition comprenant des composants qui peuvent s'exécuter en parallèle et non pas uniquement de façon séquentielle.

D'autres part, quelques travaux [(Beauche et al., 2008), (Zeng et al., 2008), (Jiang et al., 2010)] sur la composition de services Web ne prennent en considération les propriétés non-fonctionnelles qu'au moment de la sélection du meilleur plan de composition, sauf que dans certains cas, la validité ou non du plan de composition lui même dépend de ces propriétés. Dans ce sens, nous avons défini des règles de composabilité offline non-fonctionnelle qui vérifient la cohérence des propriétés non-fonctionnelles des opérations des services à composer, au stade de la construction des plans de composition, après bien sûr la vérification de leur composabilité fonctionnelle.

Par rapport à l'aspect contextuel, les travaux les plus notables dans ce sens [(Merrisa et al., 2006a), (Merrisa et al., 2006b)] se préoccupent principalement des contextes des paramètres à interconnecter. La composabilité offline contextuelle proposée par le présent travail, vérifie à la fois la cohérence des propriétés contextuelles d'ordre fonctionnel et celles d'ordre non-fonctionnel.

La conformité des schémas de données XSD des paramètres à interconnecter se présente comme une contrainte essentielle à vérifier. Deux schémas de données XSD de deux paramètres à interconnecter ne sont pas souvent équivalents syntaxiquement, mais des solutions de conversions de ces deux types de données peuvent bien exister. Dans ce

cas, notre approche propose d'exploiter les schémas de *mapping* (*LiftingSchema* et *LoweringSchema*) si une équivalence sémantique des concepts décrivant les deux paramètres est vérifiée. Le but est de vérifier automatiquement l'existence de solutions de conversions entre les types de données XSD concernés.

Par ailleurs, l'aspect technique qui n'est souvent pas pris en considération pour la composition des services Web dans la littérature, et n'est utilisé particulièrement pas lors du traitement de la composabilité des services est traité dans notre approche par la règle de composabilité offline technique. Cette règle permet notamment de vérifier la conformité des protocoles de transmission utilisés par deux opérations à interconnecter.

En conclusion, l'approche de composabilité offline proposée dans le présent chapitre est une approche qui trouve ses fondements dans les informations descriptives des services Web. Elle exploite ces informations pour traiter aux niveaux syntaxique ou sémantique, plusieurs aspects (fonctionnel, non-fonctionnel, contextuel, orienté données et technique) de composabilité des opérations de services Web, dont la plupart sont ignorés par les travaux de composition. Le but est double :

D'une part, il s'agit de dégager et identifier des opérations candidates à des collaborations à travers des plans de composition (la candidature n'est validée qu'après la satisfaction des contraintes de composabilité online qui dépendent de la requête utilisateur et son contexte).

D'autre part, le but est d'intercepter au préalable les hétérogénéités éventuelles qui peuvent impacter l'efficacité d'un plan de composition et de pouvoir les traiter en ayant recours à des solutions de médiation afin d'augmenter le nombre d'opérations à interconnecter ainsi que l'efficacité des plans de composition.

# Chapitre 6

# Dispositif de composabilité des services Web : Architecture, Algorithme et Expérimentation

## Sommaire


***6.1. Introduction ..... 151***

***6.2. Conception du dispositif ..... 151***

6.2.1. Démarche globale ..... 152

6.2.2. Base de données de comosabilité ..... 152

6.2.3. Environnement technique ..... 155

***6.3. Algorithme de composabilité offline ..... 156***

6.3.1. Fonction de composabilité fonctionnelle ..... 156

6.3.2. Fonction de composabilité contextuelle (niveau fonctionnel) ..... 157

6.3.3. Fonction de composabilité orientée données ..... 158

6.3.4. Fonction de vérification de satisfaction des politiques (composition non-fonctionnelle) ..... 159

6.3.5. Fonction de composabilité technique ..... 162

6.3.6. Vue globale de l'algorithme de composabilité offline ..... 162

***6.4. Expérimentation ..... 167***

6.4.1. Présentation des opérations candidates ..... 167

6.4.2. Présentation des scénarios de test ..... 169

6.4.3. Résultat du scénario 1 ..... 170

6.4.4. Résultat du scénario 2 ..... 172

6.4.5. Résultats du scénario 3 ..... 172

6.4.6. Contenu de la table *ComposabilityTrace* ..... 173

***6.5. Synthèse ..... 174***

## 6.1. Introduction

Dans ce chapitre, nous exposons la démarche globale adoptée pour vérifier la composabilité offline de deux opérations, aussi bien que les classes utilisées pour la mise en œuvre de notre approche. Nous présentons, ensuite notre algorithme de composabilité offline à travers les différentes fonctions de vérification de composabilité relatives à chaque niveau.

Nous avons conçu et réalisé un dispositif pour démontrer principalement la faisabilité de notre démarche. La section 6.4 illustre cette application et la valide à travers un jeu de tests.

## 6.2. Conception du dispositif

### 6.2.1. Démarche globale

La démarche globale que nous adoptons pour vérifier la composabilité offline de deux opérations, comprend trois principales phases (Figure 6.1) :

- **Extraction et stockage des informations :** cette phase consiste à extraire les informations nécessaires pour vérifier la composabilité des services Web, à partir des fichiers SAWSDL, WSDL 2.0 et WS-Policy, et à stocker ces informations dans une base (base de données de composabilité (Figure 6.1)) permettant par la suite de pouvoir les utiliser par l'algorithme de composabilité. Par ailleurs, il est à noter que seules les informations impliquées dans le traitement de la composabilité offline sont concernées. Le modèle de données est conçu dans l'optique d'optimiser les traitements de composabilité. Afin d'assurer la phase d'extraction et stockage des informations, un module JAVA a été développé en utilisant l'API EASYWSDL (cf. Annexe A).
- **Vérification de la composabilité offline :** à ce stade, on vérifie la composabilité des opérations de services Web, en utilisant notre algorithme développé en java. Cet algorithme intègre, en fait, l'ensemble des règles fonctionnelles, non-fonctionnelles, contextuelles (niveau fonctionnel et non-fonctionnel), orientées données et techniques explicitées dans le chapitre précédent.

- **Enregistrement des niveaux de composabilité :** après exécution de l'algorithme de composabilité offline, une table dite *ComposabilityTrace* (dans la base de données de composabilité) trace et mémorise les niveaux de composabilité des opérations composables offline fonctionnellement.

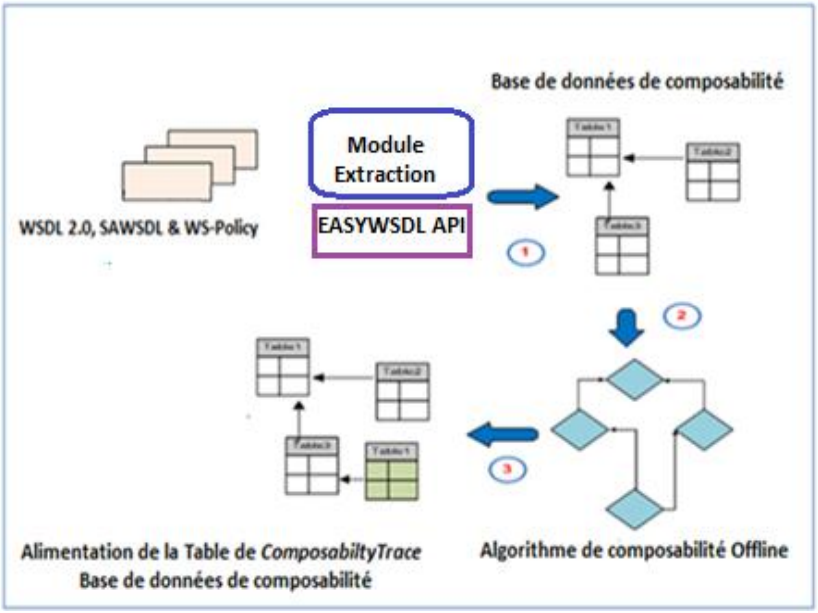


**Figure 6.1.** Démarche globale

### 6.2.2. Base de données de composabilité

Nous exposons dans la présente section, le diagramme UML de classes persistantes mises en jeu dans notre approche de composabilité offline (Figure 6.2). Ces classes sont utilisées par l'algorithme de composabilité afin, d'une part, d'identifier les opérations composables offline à des niveaux fonctionnel, non-fonctionnel, orienté données, contextuel (fonctionnel et non-fonctionnel) et technique, et d'autre part, de tracer les résultats de cette vérification.

Nous rappelons que si l'opération A, fournie par le service *S1*, est composable avec l'opération B, fournie par le service *S2*, alors *S1* est composable avec *S2*. De ce fait, l'opération fournie par un service Web se présente comme l'entité principale de notre modèle de composabilité (cf. Chapitre 5 Section 5.4).

Les classes, à l'exception de la classe spécifique *ComposabilityTrace*, sont issues des fichiers WSDL 2.0, SAWSDL et WS-Policy, et concernent précisément les concepts

appartenant à notre modèle de composabilité détaillé dans le chapitre cinq (cf. Section 5.4).

La classe ***Operation*** est définie dans notre diagramme par une clé unique *key*, une agrégation de paramètres d'entrées et de sorties ***Parameter*** et une agrégation de points d'accès ***Endpoint*** associés au service Web qui fournit l'opération en question.

La classe ***Endpoint*** est définie par l'URI de l'endpoint *uri* et le protocole de transmission *transportProtocol* prévu par la liaison « *Binding* » associée au point d'accès.

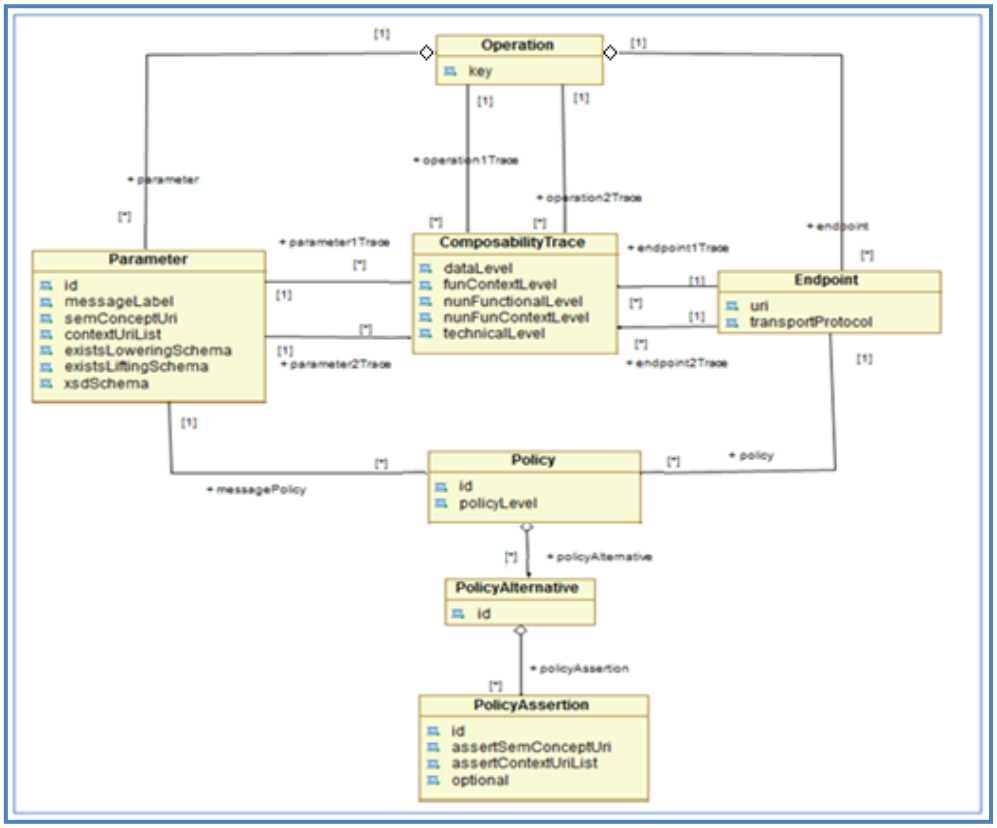


**Figure 6.2.** Diagramme UML de classes des entités persistantes mises en jeu dans la composabilité offline

La classe paramètre ***Parameter*** est définie par un identifiant unique *id*, la direction du paramètre *messageLabel*, un URI de son concept sémantique *semConceptUri*, le contenu XML de son schéma de données XSD *xsdSchema*, une liste des URIs des instances de ses concepts sémantiques contextuels c*ontextUriList*, et enfin deux attributs booléens *exitsSchemaLowering* et *existsSchemalifting*. L'attribut *messageLabel* peut

être soit *In*, soit *Out* selon qu'il s'agit respectivement d'une entrée ou d'une sortie de l'opération. Pour annoter sémantiquement le paramètre, L'URI *semConceptUri* renvoie vers le lien d'un concept ontologique. La liste des URIs c*ontextUriList* renvoie vers des instances de concepts sémantiques qui fixent les propriétés contextuelles du paramètre et permettent de l'interpréter de façon non ambiguë. Les attributs *exitsSchemaLowering* et *existsSchemalifting* renseignent respectivement sur l'existence de schéma de *mapping* de la structure de données (XSD) du paramètre vers la structure sémantique du concept qui l'annote et vice versa. L'information concernant les attributs *semConceptUri* et c*ontextUriList* est extraite de la liste des URIs renseignés par la balise *ModelReference* associée au paramètre en question au niveau du fichier SAWSDL. Quant aux valeurs des deux attributs *exitsSchemaLowering* et *existsSchemalifting*, elles peuvent être respectivement déduites suite à la vérification de l'existence ou non d'au moins un URI de schéma de *mapping* comme contenu des balises *LoweringSchema* et *LiftingSchema* utilisés pour décrire le paramètre concerné.

la classe ***Policy*** peut être attachée à un point d'accès (*Endpoint*), à la liaison (*Binding*) qui est associée à ce dernier, ainsi qu'aux service (*Service*) et opération (*Operation*) auxquels il correspond. La classe ***Policy***, et pour des raisons d'optimisation, intègre aussi les politiques de niveau *Message* attachées à un paramètre de l'opération qui est accessible via un point d'accès (*Endpoint*). Elle est spécifiée par un niveau *policyLevel* qui peut avoir comme valeur « *Endpoint », «Binding », «Service », «Operation » ou « Message »*. Par ailleurs, chaque politique ***Policy*** est une agrégation de politiques alternatives ***PolicyAlternative*** et chaque politique alternative est une agrégation d'assertions ***PolicyAssertion***. La classe ***PolicyAssertion*** est définie par l'attribut *assertSemConceptUri* qui renvoie vers le lien du concept ontologique décrivant l'assertion, l'attribut *assertContextUriList* qui renvoie vers la liste des instances de concepts sémantiques fixant les propriétés contextuelles de l'assertion, et l'attribut booléen *optional* qui permet d'indiquer si l'assertion est exigée ou est facultative.

Enfin, la classe ***ComposabilityTrace*** trace les opérations qui sont composables fonctionnellement après exécution de l'algorithme de composabilité, en renseignant les résultats des autres niveaux de composabilté relevés. Cette classe entité indique la satisfaction ou non des niveaux de composabilité (autres que le niveau de composabilité fonctionnelle) pour un uplet (*Op1, Op2, p1, p2, e1, e2*) où *Op1* et *Op2* sont des

opérations composables fonctionnellement par le biais des paramètres *p1* (entrée de l'opération *Op1*) et *p2* (sortie de l'opération *Op2*), et accessibles respectivement via les endpoints *e1* et *e2*. Aussi, est-elle décrite par les attributs booléens *nunfunctionalLevel* , *dataLevel*, *funContextLevel*, *nunfunContextLevel* et *technicalLevel* qui renseignent respectivement sur la satisfaction ou non des niveaux de composabilité non-fonctionnel, orienté données, contextuel fonctionnel, contextuel non-fonctionnel et technique.

### 6.2.3. Environnement technique

Le tableau 6.1 décrit l'environnement technique adopté pour la réalisation du dispositif de composabilité offline. Notre sélection d'outils est motivée par l'utilisation de logiciels libres et courants. Nous rappelons que l'objectif principal de la mise en œuvre de ce dispositif, dans le cadre de cette recherche est l'étude de faisabilité des mécanismes de composabilité offline conçus.

Tableau 6.1 Environnement technique

| Produit | Version | Rôle |
|---|---|---|
| EASYWSDL | 2.1 | L'API Java EasyWSDL est utilisée dans la phase d'extraction des informations. Notamment, cette bibliothèque puissante d'analyse de fichiers WSDL peut être utilisée pour analyser les descriptions WSDL 1.1, WSDL 2.0, SAWSDL et WS-Policy, et les manipuler dans un modèle objet unifié (basé sur les entités WSDL 2.0) [EASYWSDL, 2010]. |
| JDK | 1.6 | Le Kit de développement Java est utilisé pour développer le dispositif de composabilté offline. |
| MySQL Server | 5.0.27 | MySQL Server est utilisé comme serveur de base de données. |
| JBOSS | 6.0 | JBOSS est utilisé comme serveur d'application Web. |

## 6.3. Algorithme de composabilité offline

La composabilité de deux opérations est assurée par l'ensemble des règles décrites au niveau du chapitre 5. L'algorithme correspondant est conçu de façon optimisée de manière à ce que son exécution soit arrêtée une fois la règle « stricte » de composabilité n'est pas satisfaite (règle du niveau fonctionnel). En outre, l'ordre des règles de composabilité a été réfléchi en tenant compte du modèle des données de composabilité offline dans le but d'optimiser les traitements sur les entités en question.

Dans cette section, nous présentons d'abord les fonctions de composabilité offline relatives à chaque niveau, avant d'exposer une vue générique de l'algorithme global de composabilité offline qui met en avant l'ordre d'appel de chaque fonction. Cet algorithme est développé en Java 1.6 (cf. Annexe B) et présenté dans cette section en pseudo code.

Dans la suite, nous adoptons les conventions de notations suivantes : i) les opérations *Op1* et *Op2*, telles *Op1* est l'opération objet d'une vérification de composabilité avec *Op2*, ii) les paramètres *p1* et *p2*, tels, *p1* est une entrée de *Op1*, et *p2* est une sortie de *Op2*, les endpoints *e1* et *e2*, tels, *e1* est un endpoint de *Op1* et *e2* est un endpoint de *Op2*.

### 6.3.1. Fonction de composabilité fonctionnelle

La fonction *functionalComposabilityByParam* vérifie la possibilité d'interconnecter deux opérations *Op1* et *Op2* par leur biais de deux paramètres, en se basant sur leur sémantique. Elle prend en entrée trois paramètres : un paramètre *p1* entrée de l'opération *Op1*, un paramètre *p2* sortie de l'opération *Op2* et le seuil de relaxation sémantique toléré par l'utilisateur. Cette fonction retourne la valeur *true* si l'opération *Op1* peut être connectée fonctionnellement avec l'opération *Op2* par le biais des paramètres *p1* et *p2*, et *false* sinon. Les URIs des concepts sémantiques liées aux paramètres *p1* et *p2* sont en fait comparés afin de vérifier s'il existe une relation d' « équivalence » ou de « subsume » entre le concept sémantique qui décrit *p1* et celui qui décrit *p2*, ou si la distance sémantique entre ces deux concepts est inférieure au seuil toléré. Dans ces cas, l'opération *Op1* est dite fonctionnellement composable offline avec l'opération *Op2* par le biais des paramètres *p1* et *p2*.

***Boolean functionalComposabilityByParam*** *(p1:Parameter, p2: Parameter, seuil: Float)*

**Entrées** *p1*, *p2* les deux paramètres à comparer ainsi que l'entrée *seuil* qui détermine la distance sémantique tolérée

**Sortie** retourne ***true*** si l'entrée *p1* peut être connectée fonctionnellement avec la sortie *p2* et ***false*** sinon

**If** *( (p1.messageLabel == "In"* ***and*** *p2.messageLabel == "Out" )*

// Des fonctions sont utilisées pour vérifier si les concepts sémantiques de *p1* et *p2* sont // équivalents, de type Subsume ou de distance sémantique inférieure au seuil.

***and*** *(* ***semEquivalent*** *(p1.semConceptUri, p2.semConceptUri )*

***or*** ***semSubsume*** *(p1.semConceptUri, p2.semConceptUri)*

***or*** ***diSem*** *(p1.semConceptUri, p2.semConceptUri) <= seuil ) )*

**Then** ***return*** true

**Else** ***return*** false

### 6.3.2. Fonction de composabilité contextuelle (niveau fonctionnel)

La fonction *functionalContextComposability* prend comme entrée deux paramètres *p1*, entrée d'une opération *Op1*, et *p2*, sortie d'une opération *Op2*. Elle vérifie que le contexte du pramètre *p1* est couvert par le contexte du paramètre *p2*, c'est-à-dire que toutes les instances sémantiques annotant le paramètre *p1* sont incluses dans la liste des instances sémantiques annotant le paramètre *p2*. Cette fonction retourne *true* si le contexte du paramètre *p1* est couvert par le contexte du paramètre *p2*, et *false* sinon.

***Boolean functionalContextComposability*** *(p1:Parameter, p2: Parameter)*

**Entrées** *p1*, *p2* deux paramètres dont on souhaite comparer les contextes

**Sortie** retourne ***true*** si le contexte du paramètre *p1* est couvert par le contexte du paramètre *p2*, sinon ***false***

```
contextComposability:= true
iterator := p1.contextURIList.iterator()

While (iterator.hasNext() and contextComposability) Do
    If p2. contextURIList.contains( iterator.next()) == false    // instance non couverte
    Then contextComposability:= false

return contextComposability
```

### 6.3.3. Fonction de composabilité orientée données

La fonction *dataComposability* vérifie si deux paramètres *p1* et *p2* qu'elle reçoit en entrée, sont compatibles au niveau structure de données. Leur compatibilité peut être assurée par la vérification de l'équivalence syntaxique de leur schémas XSD, ou de l'existence d'un *lifting schema* associé au paramètre *p1* et d'un *lowering schema* associé au paramètre *p2* dans le cas d'une équivalence des concepts sémantiques annotant les deux paramètres. Cette fonction retourne la valeur *true* si le paramètre *p1* est compatible niveau structure de données avec le paramètre *p2*, et *false* sinon.

***Boolean dataComposability*** *(p1:Parameter, p2: Parameter)*

**Entrées** *p1*, *p2* deux paramètres dont on souhaite comparer les schémas de données

**Sortie** retourne ***true*** si les schémas de données des paramètres d'entrée sont compatibles et ***false*** sinon

```
If ( (p1.xsdSchema.getString() ==  p2.xsdSchema.getString() )
    or (  ( semEquivalent (p1.semConceptUri, p2.semConceptUri )
          and ( p1. existsLiftingSchema ==  true)
          and  (p2 . existsLoweringSchema ==  true) )
       )
```

**Then** ***return true***

**Else** ***return false***

### 6.3.4. Fonction de vérification de satisfaction des politiques (composition non-fonctionnelle)

La vérification de la satisfaction d'une politique par une autre est effectuée par la fonction *satisfy*. Cette fonction est fondée sur l'appariement sémantique des assertions exigées des deux politiques. En pratique, la politique *policy1* est dite satisfaite par la politique *policy2* s'il existe au moins une politique alternative de *policy1* qui est satisfaite par une politique alternative *policy2*. Dans ce sens, la fonction *satisfy* vérifie que chacune des assertions exigées par la politique alternative de *policy1* est annotée par un concept sémantique qui est équivalent, « subsume », ou similaire au concept sémantique annotant l'une des assertions proposées par la politique alternative de *policy2*.

Par ailleurs, comme nous l'avons noté précédemment, pour des raisons d'optimisation des traitements, la fonction *satisfy* procède à la vérification de la couverture contextuelle des assertions au fur et à mesure de l'appariement sémantique des assertions. Elle vérifie en même temps si une politique *policy1* est satisfaite par une politique *policy2* et si le contexte de *policy1* est couvert par celui de *policy2*. Dans ce sens, elle appelle la fonction *assertCxtComposability* pour vérifier si le contexte d'une assertion *a1* est couvert par celui d'une assertion *a2*. A cette fin, cette dernière vérifie si toutes les instances sémantiques annotant l'assertion *a1* sont incluses dans la liste des instances sémantiques annotant *a2*.

Finalement, la fonction *satisfy* prend en entrée les deux politiques *policy1* et *policy2* à apparier, en plus du seuil considéré pour calculer la distance sémantique. Elle retourne deux sorties, *policyComp* qui est égale à *true* si *policy2* satisfait *policy1* et *false* sinon, ainsi que *policycxtComp* qui est égale à *true* si les deux politiques ont des contextes compatibles et *false* sinon.

✓ **Fonction « *assertCxtComposability* »**

***Boolean assertCxtComposability*** *(a1:Assertion, a2: Assertion)*

**Entrées** *a1*, a*2* deux assertions dont on souhaite vérifier la cohérence de contextes

**Sortie** retourne ***true*** si le contexte de l'assertion *a1* est couvert par le contexte de l'assertion *a2*, sinon ***false***

```
assertCxtComposability:= true
iterator := a1.assertContextURIList.iterator()

    While (iterator.hasNext() and assertCxtComposability ) Do
        If a2. assertContextURIList.contains( iterator.next()) == false
        Then assertCxtComposability:= false
    return assertCxtComposability
```

✓ **Fonction « *satisfy* »**

***satisfy*** *(policy1:Policy, policy2: Policy, policySatisfied: Boolean , policycxtComp: Boolean, Float seuil )*

**Entrées** *policy1*, *policy2* deux politiques objets d'appariement ainsi que l'entrée seuil qui détermine la distance sémantique tolérée

**Sorties** la variable *policyComp* dont la valeur est égale à ***true*** si *policy2* satisfait *policy1* et ***false*** sinon, et la variable *policycxtComp* dont la valeur est égale à ***true*** si les deux politiques ont des contextes compatibles et ***false*** sinon

```
policySatisfied:= false

policyAltIterator1 := policy1.policiesAlt.iterator()
```

```
While (policyAltIterator1.hasNext() and !(policySatisfied) ) Do
    policyAltNotSatisfied := true
    policyAltIterator2 := policy2.policiesAlt.iterator()
    While (policyAltIterator2.hasNext() and policyAltNotSatisfied) Do
        lastAssertionSatisfied := true
        policycxtComp := true
        assert1 := policyAltIterator1.next().assertions.iterator()
        While (asset1.hasNext() and lastAssertionSatisfied) Do
            assertionSatisfied := false
            assert2 := policyAltIterator2.next().assertions.iterator()
                While (asset2.hasNext() and !(assertiontSatisfied) ) Do
                    If (( semEquivalent (assert1.semConceptUri, assert2.semConceptUri )
                        or semSubsume (assert1.semConceptUri, assert2.semConceptUri)
                        or diSem(assert1.semConceptUri,assert2.semConceptUri <= seuil ))
                    Then assertiontSatisfied := true
                        If (policycxtComp ) /* ne plus vérifier la cohérence du contexte des
                                Deux politiques alternatives si au moins une paire d'assertions
                                (assert1, assert2) parmi celles qui satisfont leur composabilité
                                non-fonctionnelle, ne vérifie pas la condition de cohérence
                                contextuelle */
                        Then policycxtComp := assertCxtComposability (assert1,assert2)
                    Else asset2.next()
            lastAssertionSatisfied := assertiontSatisfied
            asset1.next()
        policyAltNotSatisfied:= ! (lastAssertionSatisfied)
    policySatisfied:= policyAltNotSatisfied
```

### 6.3.5. Fonction de composabilité technique

La fonction *technicalComposability* vérifie si les protocoles de transmission associés à deux endpoints *e1* et *e2* sont similaires.

***Boolean technicalComposability*** *(e1:Endpoint, e2: Endpoint)*

**Entrées** *e1*, *e2* deux endpoints via lesquels les deux opérations, objets d'une vérification de composabilité sont accessibles

**Sortie** retourne ***true*** si les deux opérations sont composables techniquement à travers les deux endpoints *e1*, *e2*, et ***false*** sinon

**If** ( (*e1.transportProtocol.getString() == e1.transportProtocol.getString() )*

**Then** ***return true***

**Else** ***return false***

### 6.3.6. Vue globale de l'algorithme de composabilité offline

L'algorithme de composabilité offline utilise les différentes fonctions présentées précédemment dans un ordre réfléchi et séparant deux phases afin d'optimiser les traitements. Cet ordre consiste d'abord à commencer par identifier les opérations composables fonctionnellement, une condition nécessaire pour que les traitements de composabilité des autres niveaux aient raison d'avoir lieu. L'algorithme sépare également deux phases :

- ✓ La première consiste à effectuer des traitements de composabilité relatifs à chaque couple de paramètres d'interconnexion de deux opérations.

- ✓ La deuxième phase permet d'effectuer un ensemble de traitements de composabilité liés essentiellement aux endpoints par lesquels les opérations composables fonctionnellement, sont accessibles, mais qui inclut des sous-traitements relatifs aux paramètres de ces opérations.

La séparation de cette phase permet en fait d'optimiser l'algorithme, qui aurait dû comprendre des traitements (de la deuxième phase) redondants si ces derniers étaient traités pour chaque couple de paramètres de la première phase.

**Première phase**

Notamment, pour une opération *Op1* objet de vérification de composabilité offline avec une opération *Op2*, cet algorithme vérifie tout d'abord, pour chaque entrée *p1* de *Op1* s'il existe une sortie *p2* de *Op2* connectable fonctionnellement avec *p1*. Dans ce cas, il procède dans une première phase à la vérification d'une première partie des niveaux de composabilité relatifs à la cohérence contextuelle des deux paramètres (niveau fonctionnel) et celle de leurs schémas de données. Il crée également une instance de l'objet *ComposabilityTrace* afin d'y stocker les résultats obtenus et l'ajoute à une liste *list* des objets *ComposabilityTrace* permettant par la suite de repérer l'ensemble des paires de paramètres par lesquels les deux opérations *Op1* et *Op2* peuvent être connectées, ainsi que les résultats de composabilité de leurs niveaux contextuel et orientés données.

**Deuxième phase**

L'algorithme procède dans une deuxième phase, à la vérification d'une deuxième partie des niveaux de composabilité. La vérification de ces niveaux consiste en la vérification de la composabilité non-fonctionnelle, contextuelle (niveau non-fonctionnel) et technique pour chaque deux endpoints *e1* et *e2*, par lesquels sont respectivement accessibles les deux opérations *Op1* et *Op2*, et ce tout en vérifiant également la conformité des politiques de niveau message associées aux couples de paramètres identifiés dans la première phase (appartenant à la liste *list*).

**Vérification des *Policy***

Cette vérification est effectuée en deux étapes. La fonction vérifie, d'abord, pour chaque politique *policy1* associée à l'endpoint *e1*, au *Binding* associé, à l'opération *Op1* ou au Service correspondant, s'il existe une politique du même niveau *policy2* associée à *e2* et qui satisfait la politique *policy1*. Dans un tel cas, la deuxième étape consiste à vérifier, pour chaque couple de paramètres par le biais desquels les opérations *Op1* et *Op2* sont composables fonctionnellement, si toute politique de niveau message attachée au

paramètre d'entrée est satisfaite par une politique de niveau message associée au paramètre de sortie.

**Enregistrement de la trace**

Finalement, pour chaque uplet (*Op1*, *Op2*, *p1*, *p2*, e1, e2), une ligne est créée au niveau de la table *ComposabilityTrace* afin de tracer les résultats de l'ensemble des niveaux de composabilité relatifs à cet uplet, y compris les niveaux non fonctionnel, contextuel (niveau fonctionnel), orienté données et technique.

***Algorithme : ComposabilityOffline****( Operation op1,Opertaion op2, seuil Float)*

**Entrées:** *op1* et *op2* les deux opérations à vérifier leur composabilité offline

List *list*

```
// Tracer la première partie des niveaux de composabilité de deux opérations
// composables fonctionnellement par le biais de deux paramètres

Foreach p1 in op1.params Do
    Foreach p2 in op2.params Do
        If functionalComposabilityByParam (p1, p2, seuil)
        Then ComposabilityTrace t := new ComposabilityTrace (op1, op2, p1, p2)
             t. funContextLevel:= fuctionalContextComposability(p1, p2, seuil)
             t. dataLevel:= dataComposability (p1, p2)
             list. add (t)

//Tracer la deuxième partie des niveaux de composabilité de toutes les opérations
// composables fonctionnellement par le biais de deux paramètres
If list <> null
Then Foreach e1 in op1.endpoints Do
        Foreach e2 in op2.endpoints Do
            Boolean nFuncLvl:= true
            Boolean nunFuncCxtLevelFinal :=true
            Boolean nFuncCxtLvl :=true
   // Vérifier la satisfaction des politiques aux niveaux Binding, Operation, Service et Endpoint
            While (policy1 in e1.policies and nFuncLvl) Do
                While (policy2 in e2.policies and nFuncLvl) Do
                    If (policy1.level==policy2.level) & (policy1.level<>"messageLevel")
                    Then satisfy (policy1, policy 2, nFuncLvl, nFuncCxtLvl, seuil)
                     If nFuncCxtLvl == false
                     Then nunFuncCxtLevelFinal:= false
                    policy2.next()
                policy1.next()
```

```
Foreach t in list Do
// Vérifier la satisfaction des politiques aux niveau Meesage pour les paramètres fonctionnelment
// Composables
While (policy1 in getpolicies(t.p1, e1) and nFuncLvl) Do
    While (policy2 in getpolicies(t.p2, e2) and nFuncLvl) Do
        satisfy (policy1, policy 2, nFuncLvl, nFuncCxtLvl, seuil)
        If nFuncCxtLvl == false
        Then nunFuncCxtLevelFinal:= false
        policy2.next()
    policy1.next()
    If nFuncLvl == false // si le niveau non-fonctionnel n'est pas satisfait le niveau
                         // Contextuel est automatiquement insatisfait
    Then nunFuncCxtLevelFinal:= false
    // Stocker les informations concernant l'uplet (op1, op2, p1, p2, e1, e2)
    ComposabilityTrace v := new ComposabilityTrace ( t )
    v.endpoint2:= e2.uri;
    v.endpoint2:= e2.uri;
    v. nunfunctionalLevel := nFuncLevel
    v. nunfunContextLevel := nunFuncCxtLevelFinal
    v. technicalLevel := technicalComposability(e1, e2)
    v.persist();
```

## 6.4. Expérimentation

La partie expérimentation vise à mieux illustrer la démarche globale adoptée pour vérifier la composabilité offline de deux opérations et expliciter ses différentes étapes. Pour réaliser cette partie, nous avons pris le cas de trois opérations *getBookPrice* , *ePay* et *BookShip* appartenant respectivement à trois services Web *« BookShop »*, *« Payement »* et *« Shipment »*, afin de traiter leur composabilité offline à partir des données extraites des fichiers WSDL 2.0 des trois services développés dans le cadre du présent travail.

### 6.4.1. Présentation des opérations candidates

Le tableau 6.2 ci-dessous présente trois opérations *getBookPrice* , *ePay* et *BookShip* utilisées dans les scénarios de tests réalisés. Les informations descriptives concernant ces trois opérations sont extraites des fichiers WSDL 2.0 correspondants. Afin d'exposer ces informations conformément au diagramme de classes UML des entités persistantes utilisées pour la mise en œuvre de notre approche de composabilité offline, nous présentons les diagrammes d'objets relatifs à chaque opération.

Tableau 6.2 Descriptions des opérations utilisés dans les scénarios de test

| Clé | Nom | Rôle |
|---|---|---|
| 100 | *getBookPrice* | L'opération *getBookPrice* permet d'avoir le prix d'un livre à partir de son ISBN. Elle prend en entrée le paramètre *BookISBN*, et envoie comme sortie le paramètre *BookPrice* (figure 6.3). |
| 200 | *ePay* | L'opération *ePay* permet de payer électroniquement un montant donné en utilisant la carte de crédit. Elle prend en entrée les paramètres *Price* et *NumCreditCard*, et envoie en sortie un identifiant du paiement effectué *PaymentID* (figure 6.5). |

| | | |
|---|---|---|
| 300 | BookShip | L'opération BookShip informe sur la date de livraison du livre acheté sur la base de sa référence paiement. Elle prend en entrée le paramètre *PaymentRef*, et envoie le paramètre *ShipmentDate* (figure 6.4). |

L'opération *getBookPrice* peut être accédée à travers deux endpoints *HttpBookPrice* et *SoapBookPrice*. *HttpBookPrice* est accessible à travers l'URI http://www.bookprice.com/rest/ et adopte le protocole de transport HTTP. Quant à *SoapBookPrice*, il est accessible à travers l'URI http://bookprice.com/soap/ et adopte le protocole de transport SOAP (Figure 6.3).

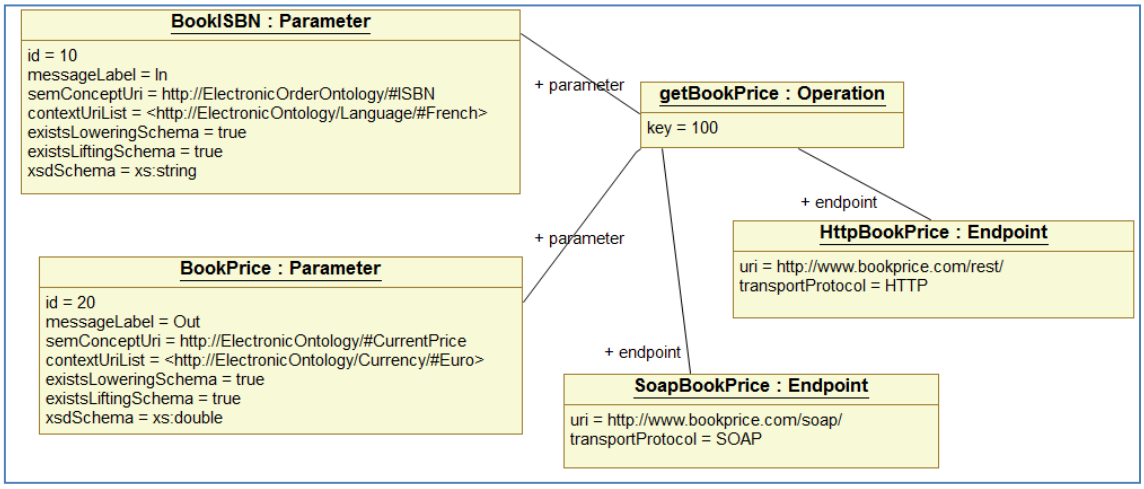


**Figure 6.3.** Diagramme d'objets de l'opération *getBookPrice*

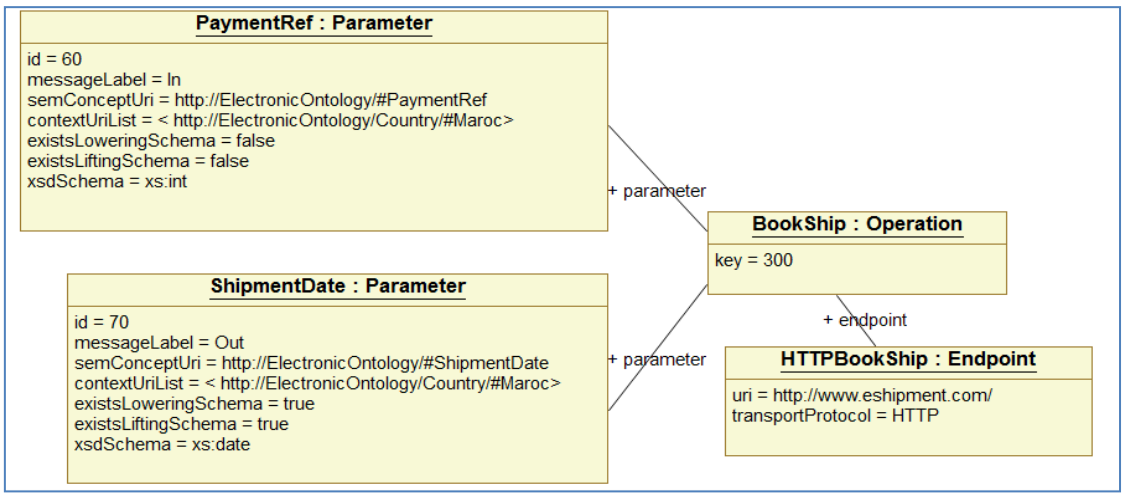


**Figure 6.4.** Diagramme d'objets de l'opération *BookShip*

Le diagramme d'objets (Figure 6.5) modélise l'opération *ePay*. La politique de sécurité *Crypted* de niveau *Message* est appliquée à l'entrée Price et la sortie *NumCreditCard* de l'opération *ePay* (Figure 6.5). Cette politique contient une politique alternative *CryptedAlt*.

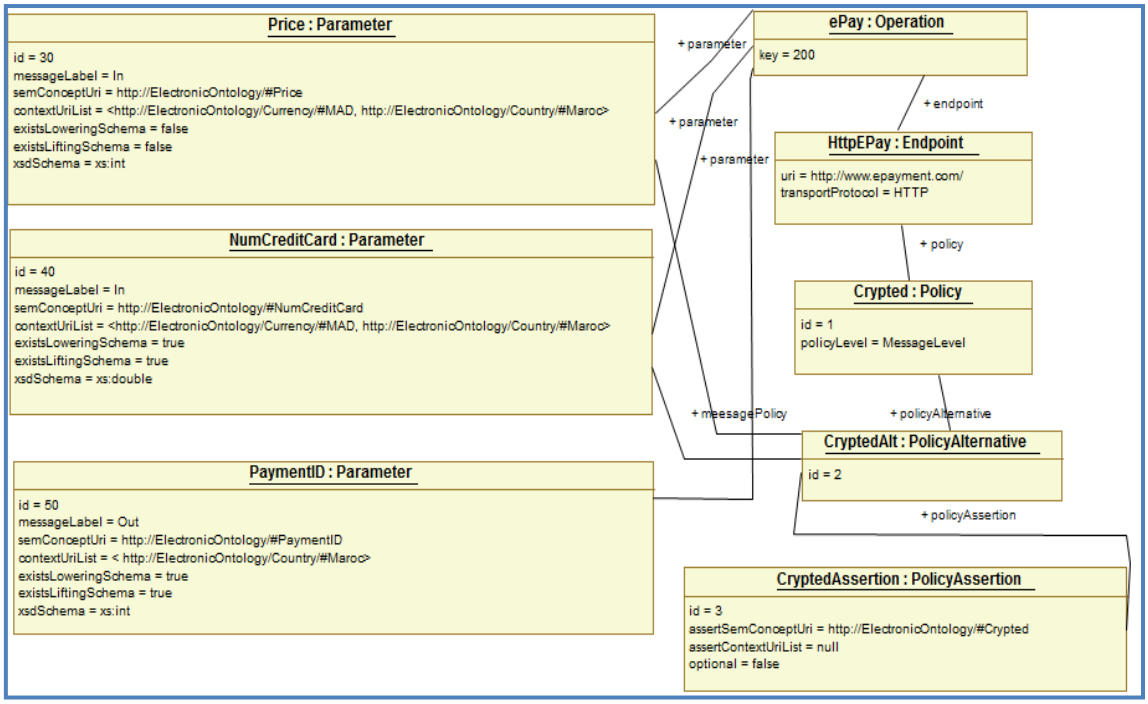


**Figure 6.5.** Diagramme d'objets de l'opération *ePay*

### 6.4.2. Présentation des scénarios de test

Pour tester l'algorithme de composabilité développé, nous avons utilisé trois scénarios définis dans le tableau 6.3 ci-dessous. Le but était d'examiner les résultats obtenus de vérification de la composabilité offline de trois paires d'opérations, tout en considérant un échantillon assez diversifié de cas à traiter par l'algorithme.

Tableau 6.3 Scénarios de test

| Scénario | Opération 1 | Opération 2 |
|---|---|---|
| Scénario 1 | ePay | getBookPrice |
| Scénario 2 | getBookPrice | ePay |
| Scénario 3 | BookShip | ePay |

Nous notons que les concepts sémantiques (identifiés par leurs URIs) impliqués dans le traitement des trois scénarios étudiés, entretiennent des relations sémantiques que nous exposons dans le tableau 6.4. Ces relations sont considérées par notre algorithme lors de la vérification des différents niveaux de composabilité.

Tableau 6.4 Relations sémantiques entre des concepts ontologiques

| Uri du Concept 1 | Uri du Concept 2 | Relation sémantique |
|---|---|---|
| http://ElectronicOntology/#Price | http://ElectronicOntology/#CurrentPrice | Subsumes |
| http://ElectronicOntology/#BookISBN | http://ElectronicOntology/#PaymentID | Nulle |
| http://ElectronicOntology/#PaymentRef | http://ElectronicOntology/#PaymentID | Distance sémantique = 0.9 |

Pour le recueil des résultats d'exécution des scénarios de test évoqués auparavant, nous avons développé une interface Web. Cet interface renseigne, pour chaque deux opérations concernées *Op1* et *Op2* qui sont identifiés composables fonctionnellement, les uplets (*p1*, *p2*, *e1*, *e2*), ainsi que les niveaux de composabilité offline correspondant à chaque uplet. Nous rappelons que *p1* et *p2* sont des paramètres d'interconnexion des opérations *Op1* et *Op2*, et *e1* et *e2* sont des points d'accès à ces opérations. Dans la suite de ce chapitre, nous présentons les résultats obtenus pour chaque scénario ainsi que le contenu de la table correspondante *ComposabilityTrace*.

### 6.4.3. Résultat du scénario 1

Selon les propriétés des opérations *ePay* et *getBookPrice* décrites auparavant, ces deux opérations sont composables offline fonctionnellement par le biais des deux paramètres d'identificateurs 30 et 20, correspondant respectivement aux paramètres *Price* et *BookPrice*.

Concernant le premier uplet (30, 20, www.epayment.com, www.bookprice.com/rest/), les contextes fonctionnels des deux paramètres ne sont pas compatibles. Notamment, la devise liée au paramètre *Price* (d'identificateur 30) est *MAD*, par contre celle liée au paramètre *BookPrice* (d'identificateur 20) est *Euro*. Par ailleurs, le paramètre *Price* est aussi associé à une politique qui exige que l'information en entrée soit cryptée. Cette

exigence n'est pas satisfaite par l'information non-fonctionnelle associée au paramètre *BookPrice*. Aussi, le contexte non-fonctionnel des deux paramètres n'est-il pas compatible.

Comme illustré par la figure 6.6, l'algorithme testé a bien détecté l'ensemble des incohérences notées au niveau de ce paragraphe. L'attribut booléen *false* (désigné par la couleur rouge sur l'écran du résultat) a été affecté aux niveaux *Functional Context Level*, *Nun-Functional Level* et *Nun-Functional Context Level*.

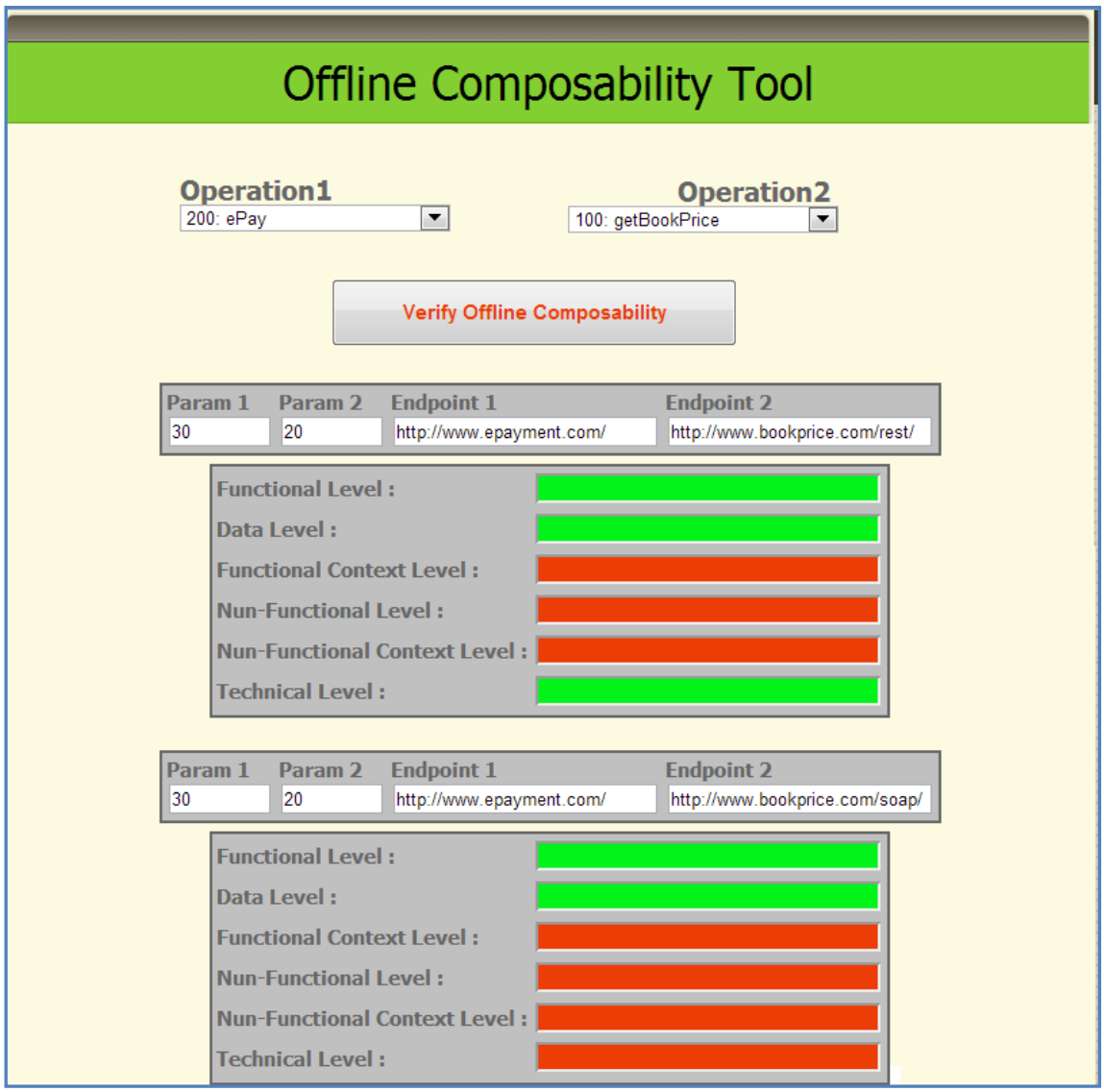


**Figure 6.6.** Ecran du résultat de composabilité offline du Scénario 1

Concernant le deuxième uplet (30, 20, www.epayment.com, www.bookprice.com/soap/), un nouveau type d'incohérence a été détecté, en plus de

ceux détectés dans le cas du premier uplet. Cette nouvelle incohérence est liée au Protocole de transport SOAP associé à l'endpoint www.bookprice.com/soap/ qui n'est pas compatible avec le protocole http, adopté par l'endpoint www.epayment.com.

### 6.4.4. Résultat du scénario 2

Selon les propriétés des opérations *getBookPrice* et *ePay* décrites auparavant, ces deux opérations ne sont pas composables offline fonctionnellement, puisqu'il n'existe aucune entrée de l'opération *getBookPrice* dont la sémantique est cohérente avec celle d'une sortie de l'opération *ePay*. Ce constat a été bien relevé par l'algorithme testé comme illustré par la figure 6.7.

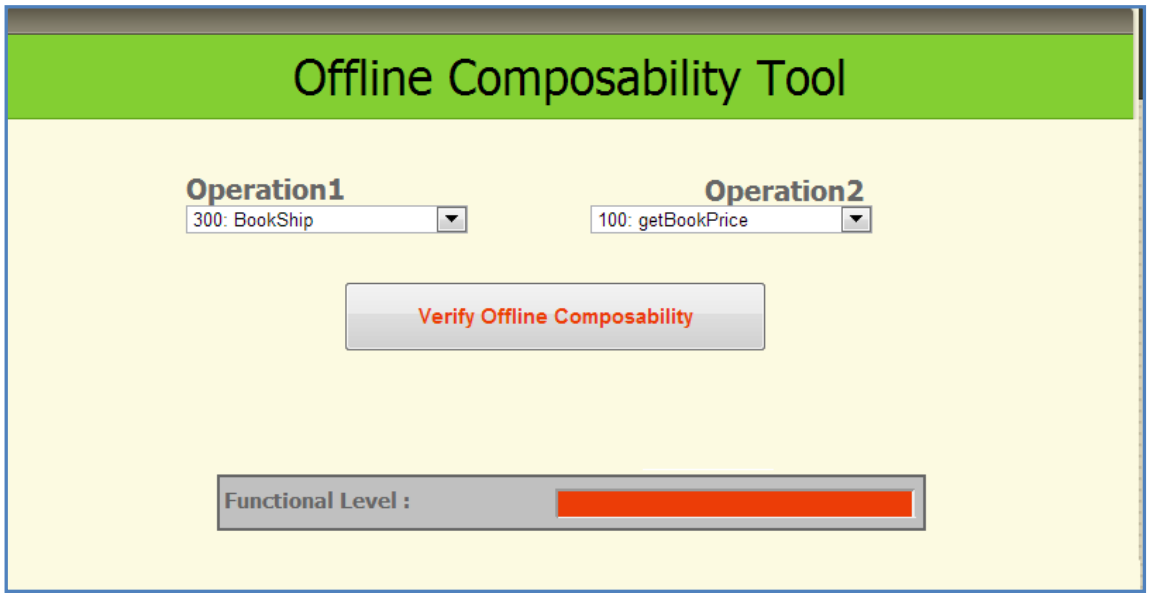


**Figure 6.7.** Ecran du résultat de composabilité offline du Scénario 2

### 6.4.5. Résultats du scénario 3

Selon les propriétés des opérations *BookShip* et *ePay* décrites auparavant, ces deux opérations sont composables offline fonctionnellement par le biais des deux paramètres d'identificateurs 60 et 50, correspondant respectivement aux paramètres *PaymentRef* et *PaymentID*. Elles sont également composables à tous les niveaux.

Comme illustré par la figure 6.8, l'algorithme testé a bien abouti aux résultats attendus. Aucune incohérence par rapport à l'ensemble des niveaux de composabilité offline n'a été détectée.


**Figure 6.8.** Ecran du résultat de composabilité offline du Scénario 3

### 6.4.6. Contenu de la table *ComposabilityTrace*

L'exécution de l'algorithme de composabilité offline élaboré débouche sur l'alimentation de la table *ComposabilityTrace*. Cette table contient des informations portant uniquement sur les opérations composables fonctionnellement. Ces informations concernent les uplets impliqués dans la composabilité des opérations identifiées ainsi que leurs niveaux de composabilité.

Le tableau 6.5 ci-dessous présente les informations qui ont été stockées au niveau de la table *ComposabilityTrace* après exécution des trois scénarios de test étudiés.

| | | | |
|---|---|---|---|
| **key Operation1** | 200 | 200 | 300 |
| **key Operation2** | 100 | 100 | 200 |
| **id Parameter1** | 30 | 30 | 60 |
| **id Prameter2** | 20 | 20 | 50 |
| **uri Endpoint1** | www.epayment.com | www.epayment.com | www.eshipment.com |

| | | | |
|---|---|---|---|
| **uri Endpoint2** | www.bookprice.com/rest/ | www.bookprice.com/soap/ | www.epayment.com |
| **dataLevel** | true | true | true |
| **funContextLevel** | false | false | true |
| **nunFunctionalLevel** | false | false | true |
| **nunFunContextLevel** | false | false | true |
| **technicalLevel** | true | false | true |

Tableau 6.5 Contenu de la table *ComposabilityTrace*

## 6.5. Synthèse

Ce chapitre expose l'approche adoptée pour la mise en œuvre des mécanismes de vérification de la composabilité offline proposés. Les propriétés descriptives des services Web, impliquées dans le processus de composabilité offline, sont d'abord extraites à partir de fichiers WSDL 2.0, SAWSDL et WS-Policy, puis chargées dans une base de données conformément au modèle de données présenté dans ce chapitre.

L'algorithme développé traite ces données dans le but d'identifier les opérations composables offline fonctionnellement et de tracer pour chaque deux opérations *Op1* et *Op2* composables offline fonctionnellement, par le biais respectivement de deux paramètres *p1* et *p2*, et accessibles respectivement via deux endpoints *e1* et *e2*, les niveaux de composabilité offline satisfaits. Ces niveaux sont relatifs aux aspects contextuel de point de vue fonctionnel, orienté données, non-fonctionnel, contextuel de point de vue non-fonctionnel et technique. L'objectif d'une telle démarche est d'une part d'identifier les opérations dont la composition donne lieu à un plan de composition exécutable, et d'autre part, de renseigner sur les types d'hétérogénéités d'opérations identifiées dans la phase de composabilté offline pour pouvoir y remédier, si possible, à travers des solutions de médiation, et par la suite pouvoir composer ces opérations sans que le service composite résultant ne soit inexécutable.

L'algorithme développé a été conçu dans une perspective d'optimiser les traitements, et ce en évitant la redondance des traitements d'une part, et d'autre part, en éliminant les traitements rendus inutiles lorsque l'information identifiée sur un niveau composabilité le stipule.

Nous avons expérimenté notre algorithme développé dans le cadre de ce travail. Cette partie avait comme objectif fondamental l'illustration globale de l'utilisation des mécanismes de composabilité offline et le test de l'algorithme conçu, à travers trois scénarios choisis pour couvrir un échantillon assez diversifié de cas de composabilité.

# Chapitre 7

# Conclusion

## Sommaire

*7.1. Contributions majeures ..... 177*

*7.2. Perspectives ..... 180*

## 7.1. Contributions majeures

L'originalité du travail de recherche mené lors de cette thèse se manifeste essentiellement dans les niveaux de détails investis aussi bien que les solutions (fondées sur une description de services alignée avec les standards W3C) proposées pour traiter les différentes facettes de la problématique de composabilité de services Web . La présente section résume les contributions majeures de notre approche :

**Etude comparative des approches de description de services Web** [Omrana et al., 2013b] : une analyse comparative fine des propriétés descriptives des services Web fournies d'un côté, par les standards W3C (WSDL 2.0, SAWSDL et WS-Policy 1.5) et d'un autre côté, par les deux modèles sémantiques OWL-S et WSMO a été élaborée dans le cadre de cette recherche.

**Approche de description des services Web fondée sur les standards WSDL 2.0, SAWSDL et WS-Policy 1.5** [(Belouadha et al., 2010b), (Omrana et al., 2011a), (Omrana et al., 2011b), (Omrana et al., 2013a), (Omrana et al., 2013b)] **:**
A l'issue d'une étude approfondie des grammaires des trois standards W3C (WSDL 2.0, SAWSDL et WS-Policy 1.5), ainsi qu'une analyse détaillée des deux modèles sémantiques OWL-S, WSMO, nous avons élaboré un modèle de description de services Web dans le but de couvrir les cinq propriétés de services : fonctionnelles, non-fonctionnelles, contextuelles, orientées données et techniques. Ce modèle UML enrichit les propriétés descriptives procurées par ces standards, par de nouvelles propriétés. En effet, nous avons exploité la flexibilité offerte par les mécanismes d'annotation sémantique de SAWSDL pour pouvoir décrire les propriétés contextuelles des messages de services échangés, et ce dans le but de permettre leur interprétation correcte. La solution proposée permet d'associer à chaque paramètre à travers son type, un ensemble d'instances qui renseignent sur son contexte fonctionnel et dont la sémantique est définie au niveau d'une ontologie.

Par ailleurs, l'exécution d'une opération peut être conditionnée par des pré-conditions à satisfaire, des post-conditions ou des effets à vérifier ou toute autre contrainte liée à son exécution. En effet, outre le concept sémantique décrivant son but fonctionnel, chaque

opération d'un service Web est annotée également, dans notre approche, par des instances sémantiques renseignant sur l'ensemble des conditions qui définissent le contexte de son exécution. D'autre part, et afin de permettre d'interpréter correctement des propriétés non-fonctionnelles exprimées par des assertions décrites en utilisant WS-Policy, nous avons proposé d'exploiter le mécanisme d'annotation de SAWSDL pour annoter sémantiquement les schémas XSD de définition desdites assertions. Cela nous a permis de décrire d'une part, la sémantique des assertions traduisant des propriétés non-fonctionnelles des services Web, et d'autre part, leurs propriétés contextuelles. Finalement, il est à noter, que l'objectif de cette approche est de capturer le maximum d'information descriptive du service Web qui peut être exploitée au niveau de la découverte ou la sélection de services Web ou la recherche de services similaires et non seulement la composabilité de services Web.

**Modèle de composabilité offline multi-aspects de services Web** [(Belouadha et al., 2010a), (Omrana et al., 2012a), (Omrana et al., 2012c)] :
Le modèle de composabilité offline de services Web proposé est fondé sur l'identification, parmi les informations descriptives des services Web recensées, de celles considérées cruciales pour la connexion de deux opérations. Ce modèle exploite ces informations pour traiter aux niveaux syntaxique et sémantique, plusieurs aspects (fonctionnel, non-fonctionnel, contextuel, orienté données et technique) de composabilité des opérations de services Web à travers les six règles de composabilité définies.

Nous avons proposé d'introduire les notions de règles « stricte» et « souple » pour évaluer l'impact de la non satisfaction d'une règle. Ainsi, la non satisfaction d'une règle « stricte» (correspondant au cas de la règle de composabilité offline fonctionnelle) élimine toute possibilité d'interconnecter deux opérations. Par contre, la non satisfaction d'une règle « souple » n'empêche pas la possibilité d'interconnecter deux opérations si une éventuelle solution de médiation manuelle, semi-automatique ou automatique peut être exploitée.

En outre, notre modèle de composabilité définit des règles de composabilité offline non-fonctionnelle et techniques qui sont rarement pris en considération lors de la vérification

de la composabilité de deux opérations. Il traite l'aspect contextuel en vérifiant la cohérence des propriétés contextuelles d'ordre fonctionnel et aussi d'ordre non-fonctionnel. Concernant, la conformité des schémas de données XSD des paramètres à interconnecter, outre leur appariement syntaxique, notre approche exploite des schémas de mapping (*LiftingSchema* et *LoweringSchema*) dans le cas d'une équivalence sémantique des concepts décrivant ces paramètres. Le but est de vérifier automatiquement l'existence de solutions de conversions entre les types de données XSD concernés.

**Contribution à la construction de plans de compositions efficaces** (Omrana et al., 2013c)**:**

Sur la base du modèle et règles de composabilité offline proposés, nous avons proposé une démarche globale pour la vérification automatique des aspects de composabilité offline de deux opérations. La démarche se compose de trois phases distinctes :

i) extraction et stockage des informations : cette phase consiste à extraire les informations nécessaires pour vérifier la composabilité des services Web, à partir des fichiers SAWSDL, WSDL 2.0 et WS-Policy, et à stocker ces informations dans des tables permettant par la suite de pouvoir les utiliser par l'algorithme de composabilité ;

ii) vérification de la composabilité : cette phase procède à la vérification de la composabilité des opérations de services Web. Elle est prise en charge par un algorithme que nous avons développé à cette fin, et qui intègre l'ensemble des règles fonctionnelles, non-fonctionnelles, contextuelles (niveau fonctionnel et non-fonctionnel), orientées données et techniques exposées ;

iii) enregistrement des niveaux de composabilité : cette phase trace et persiste les différents niveaux de composabilité des opérations composables offline fonctionnellement, dans une table créée à cette fin. L'objectif de notre démarche est d'une part, de permettre d'identifier les opérations dont la composition peut donner lieu à un plan efficace, en spécifiant avec précision les paramètres connectables, ainsi que les endpoints qui assurent cette composabilité. D'autre part, il renseigne sur les types d'hétérogénéités

d'opérations, identifiées dans la phase de composabilité offline pour pouvoir y remédier si possible à travers des solutions de médiation. Cet algorithme a été développé et expérimenté dans le cadre de cette recherche, pour illustrer l'utilisation des mécanismes de composabilité offline élaborées.

## 7.2. Perspectives

Nous estimons que la présente contribution constitue une base de travail pour plusieurs perspectives connexes envisagées actuellement dans le cadre des travaux de recherche menés par notre équipe. En fait, le modèle de description de services Web élaboré offre une masse d'information importante qui peut être fondamentale pour automatiser efficacement des processus liés aux services Web tels que la découverte, la recherche de services similaires, la composition, etc. Dans la continuité de nos recherches nous dégageons en particulier les perspectives suivantes.

**Le traitement de l'aspect comportemental dans la composabilité de services Web -** se présente comme une piste de recherche nécessaire qui complète notre thèse. Il nous semble intéressant de traiter l'aspect comportemental en exploitant les fichiers du standard OASIS BPEL4WS (orchestration de services) et le standard W3C WS-CDL (chorégraphie de services). En d'autres termes, vérifier la composabilité de deux services composites d'un point de vue comportemental, en tenant compte de la description comportementale spécifiée par lesdits standards.

**L'extension de l'algorithme de composabilité offline** – nous proposons d'étendre le présent algorithme de façon à pouvoir tracer, avec précision, l'ensemble des éléments descriptifs objets d'hétérogénéités (assertion, propriété contextuelle liée à une entrée, etc.). Actuellement, notre algorithme détecte les éventuelles hétérogénéités mais informe globalement sur le niveau de composabilité (satisfait ou non). Le but de cette perspective est d'identifier et garder trace des éléments descriptifs qui affectent un niveau de composabilité donné entre deux opérations. Cela permettrait de les repérer dans le cas d'élaboration de solutions de médiations automatiques ou semi-automatiques. Une optimisation de l'algorithme développé peut aussi être considérée.

**L'élaboration d'un modèle de notation dynamique –** cette perspective vise à doter notre modèle de composabilité offline d'un modèle de notation dynamique qui permet de classer les résultats de composabilité offline obtenus selon leur degré d'efficience. Par exemple, les relations sémantiques équivalentes devraient être mieux notées que celles de type « Subsume » ou celles dont la distance sémantique est tolérable. Dans le même sens, les opérations totalement composables devaient être privilégiées dans le système de notation par rapport à celles partiellement composables. Enfin, certaines propriétés non-fonctionnelles, peuvent êtres fixées à l'avance et renseignées afin d'être exploitées dans la classification de ces résultats. Il reste à noter que la classification est intéressante dans le cas des opérations composables offline à tous les niveaux pour choisir le meilleur plan de composition, mais aussi nécessaire dans le cas des opérations dont certains niveaux de composabilité sont insatisfaits dans le but d'optimiser les choix des solutions de médiation à développer.

# Bibliographie


Akkiraju, R., Farrell, 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.

Akkiraju, R., Sapkota, B., Semantic Annotations for WSDL and XML Schema — Usage Guide, 2007. Available at: http://www.w3.org/2002/ws/sawsdl/spec/examples/

Alonso, G., Casati, F., Kuno, H. and Machiraju, V. Web Services: Concepts, Architectures and Applications. DataCentric Systems and Applications. Springer, Heidelberg/New York, 2004.

Alves et al., Web Services Business Process Execution Language Version 2.0. OASIS Standard, April 2007. Available at http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D., McIlraith, S., Narayanan, S., Paolucci, M., Payne, T., Sycara, K., Zheng, H. : Daml-s semantic markup for web services. In Proceedings of International Semantic Web Conference (ISWC), Sardinia, Italy, 2003, pages: 348–364.

Araujo, J., Moreira, A., Brito, I., et Rashid., A., Aspect-oriented Requirements with UML, Workshop on Aspect-Oriented Modeling with UML (held with UML 2002).

Ardagna, D., Comuzzi, M., Mussi, E., Pernici, B., et Plebani, P., Paws: A framework for executing adaptive web-service processes. IEEE Software, 24, pages:39–46, 2007.

Arkin, A. et al., XML Schema Part 0: Primer Second Edition. Technical report, World Wide Web Consortium (W3C), October 2004.

Arkin, A. et al.,Web Service Choreography Interface (WSCI) 1.0: W3C Note 8 August 2002. Available at: http://www.w3.org/TR/wsci/

Azmeh, Z., Driss, M., Hamoui, F., Huchard, M., Moha, N., Tibermacine, C., Selection of Composable Web Services Driven by User Requirements, Web Services (ICWS), 2011 IEEE International Conference, pp.395,402, July 2011

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.

Baltá, A., Fernández, A., Web Service Description Alignment. Proceeding of the thirth international Symposiom on Web Services. Pages 212-218. Dubai, UAE, April 2012.

Barros A., Dumas M., Oaks P., « A Critical Overview of the Web Services Choreography Description Language (WS-CDL) », Proc. of the Business Process Trends (BPTrends), 2005

Battle, S., Bernstein, A., Boley H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith S., McGuinness D., Su J., and Tabet S.. Semantic Web Services Framework (SWSF) Overview. World Wide Web Consortium, Member Submission SUBM-SWSF-20050909, September 2005

Beauche, S. et Poizat, P. , Automated service composition with adaptive planning. ICSOC, volume 5364 of Lecture Notes in Computer Science, 2008, pages 530–537.

Beckett, D., McBride, B., RDF/XML Syntax Specification, W3C Recommendation 10 February 2004. Available at : http://www.w3.org/TR/rdf-syntax-grammar/

Bellwood, T., et al.. UDDI version 2.0.4, July 2002.

Bellwood, T., et al.. UDDI version 3.0.2, October 2004.

Belouadha, F.Z., Omrana, H., et Roudiès, O., A MDA approach for defining WS-Policy semantic non-functional properties. International Journal of Engineering Science and Technology (IJEST), Volume 2, Issue 6, 2010c, pages: 2164-2171. Avalibale at: http://www.ijest.info/docs/IJEST10-02-06-115.pdf.

Belouadha, F.Z., Omrana, H., et Roudiès, O., Web services-enhanced agile modeling and integrating business processes. E-Business - Applications and Global Acceptance, Publisher: InTech, February 2012b, pages: 73-98

Belouadha, F.Z., Omrana, H., et Roudiès, O., A model-driven approach for composing SAWSDL semantic Web services. IJCSI international journal, Volume 7, issue 2, 2010a, pages : 7-15. Avalibale at: http://www.ijcsi.org/papers/7-2-1-7-15.pdf.

Benatallah, B., Dijkman, R. and Dumas, M., In Service- Oriented Software Engineering: Challenges and Practices(Eds, Stojanovic, Z. and Dahanayake, A.) Idea Group Inc (IGI), 2005a, pages. 48-66

Benatallah, B., Dumas, M., Fauvet, M.-C., and Rabhi, F., Patterns and skeletons for parallel and distributed computing, chapter Towards Patterns of Web Services Composition, pages 265–296, 2003b.

Benatallah, B., Hacid, M., Leger, A., Rey, C. et F Toumani, F., On automating web services discovery. The VLDB Journal, 14(1), pages :84–96, 2005b.

Benatallah, B., Hacid, M., Rey, C. et F Toumani, F., Request rewriting-based web service discovery, 2003a.

Berners-Lee, T., Hendler, J. et Lassila, o., The Semantic Web, In Scientific American, May 2001, pages: 35-43.

Birukou, A. et al. , Improving Web Service Discovery with Usage Data. IEEE SOFTWARE, November/December 2007

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C. et Orchard, D., Web services architecture, 2004

Bosca, A., Ferrato, A., Corno, D.., Congui, I., and Valetto, G., Composing Web Services on the Basis of Natural Language Requests‖, Proceedings of the 3rd IEEE International Conference on Web Services (ICWS 2005),pp. 817-818, Orlando, Fl, June 2005.

Bruijn, J., et al. , The Web Service Modeling Language WSML. WSML Final Draft, 2005, Available at: http://www.wsmo.org/TR/d16/d16.1/v0.21/

Casati, F. et Shan, M.-C., Event-Based Interaction Management for Composite EServices in eFlow, Information Systems Frontiers, 4(1), 2002, pages: 19-31.

Casati, F., Ilnicki, S. et Jin. L., Adaptive and dynamic service composition in EFlow. In Proceedings of 12th International Conference on Advanced Information Systems Engineering (CAiSE), Stockholm, Sweden, June 2000.

Casati, F., Sayal, M., et Shan. M.-C., Developing e-services for composing eservices. In Proceedings of 13th International Conference on Advanced Information Systems Engineering (CAiSE), Interlaken, Switzerland, June 2001.

Ceri, S., Comai, S., Damiani, E., Fraternali, P., Paraboschi S. et Tanca, L., XML-GL : a graphical language for querying and restructuring XML documents‖, 8th WWW conference, Toronto, Canada. journal of web semantics, 1999.

Chinnici, R., Moreau J. J., Ryman A., and Weerawarana S., Web Services Description Language (WSDL) Version 1.2, 2002. Available at:: http://www.w3.org/TR/wsdl12.

Chinnici, R., Moreau J. J., Ryman A., and Weerawarana S., Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C recommendation, 2007. Available at:: http://www.w3.org/TR/wsdl20/

Chung, L., Nixon, B.A., Yu, E. and Mylopoulos, J., Non-Functional Requirements in Software Engineering, KluwerAcademic Publishers, Boston Hardbound, October 1999.

Clark, J., XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16 November 1999. Available at : http://www.w3.org/TR/xslt

Cremene, M., Tigli, J., Lavirotte, S., Pop, F., Riveill, M., et Rey, G. , Service Composition Based on Natural Language Requests, in Proc. IEEE SCC, 2009, pages.486-489.

EASYWSDL, 2010. Available at: http://easywsdl.ow2.org/

Ernst, M. D., Lencevicius, R., Perkins, J. H., Detection of Web Service substitutability and composability, International Workshop on Web Services Modeling and Testing (WS-MaTe 2006), pages: 2-14, 2006

Farrell, J. and H. Lausen. Semantic Annotations for WSDL and XML Schema, W3C recommendation, 2007. Available at: http://www.w3.org/TR/sawsdl/

Fensel, D., Bussler, C., The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications 1, 2002. Pages: 113–137

Fikes, R., Hayes , P. et Horrocks, I. , OWL-QL—a language for deductive query answering on the Semantic Web‖, Technical Report KSL-03-14, Knowledge Systems Lab, Stanford University, CA, USA, 2003.

Firat. A., Information Integration Using Contextual Knowledge and Ontology Merging. PhD thesis, Massachusetts Institute of Technology, Sloan School of Management, 2003.

Foster, H., Uchitel, S., Magee, J. et Kramer, J., Model-based Verification of Web Service Composition, October 2003.

Fujii, K., et Suda., T., Semantics-based context-aware dynamic service composition. TAAS, 4(2), 2009.

Gekas, J. et Fasli, M., Automatic Web Service Composition Based on Graph Network Analysis Metrics, In Proceedings of the International Conference on Ontology, Databases and Applications of Semantics (ODBASE). Agia Napa, Cyprus, 2005, pages: 1571-1587.

Goh, C. H., Bressan, S., Madnick, S. E. et Siegel., M., Context interchange : New features and formalisms for the intelligent integration of information. ACM Trans. Inf. Syst., 17(3), 1999, pages:270–293

Hashemian, S.V.; Mavaddat, F., A graph-based approach to Web services composition, Applications and the Internet. Proceedings. The 2005 Symposium on 2005, pages: 183,189

Hatzi, O., Meditskos, G., Vrakas, D., Bassiliades, N., Anagnostopoulos, D., Vlahavas, I., Semantic Web Service Composition using Planning and Ontology Concept Relevance with PORSCE II, The IEEE / WIC / ACM Conference on Web Intelligence, 2009.

Hatzi, O., Vrakas, D., Nikolaidou, M., Bassiliades, N.,, An Integrated Approach to Automated Semantic Web Service Composition through Planning. IEEE TRANSACTIONS ON SERVICES COMPUTING, 2011

Hong-jie, G., Fan-rong, M., Jin-fei, S. et Peijun, D., Web Service Discovery Based on the Cooperation of UDDI and DF. Proceedings of the 4th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM '08), pages. 1-4, 2008.

IBM, Getting Started: WebSphere Process Server and WebSphere Enterprise Service Bus V6, 2007. Available at: http://www.redbooks.ibm.com/abstracts/sg247378.html?Open

Intalio and BPMI, Bussiness process modeling language, 2002, Available at: http://www.bpmi.org/bpmi-downloads/BPML-SPEC-1.0.zip

Intalio, Documentation: Everything you need to get started, 2013. Available at: http://bpms.intalio.com/getting-started/10-first-steps-with-intalio-bpms.html

Iwasa, K., et al., WS-Reliability 1.1, OASIS Standard, 15 November 2004. . Available at : http://docs.oasis-open.org/wsrm/ws-reliability/v1.1

Ji, X. (2009). "Research on Web Service Discovery Based on Domain Ontology." Proceedings of the 2nd IEEE International Conference on Computer Science and Information Technology (iccsit), pages: 65-68, 2009.

Jiang, W., Zhang, C., Huang, Zh., Chen, M., Hu, S., Liu, Zh., Qsynth: A Tool for QoS-Aware Automatic Service Composition. In Int. Conf. on Web Services, 2010, pages: 42–49

Juric, M. B., Business Process Execution Language for Web Services Second Edition, Packt Publishing Ltd, 2006

Karakoc, E., et Senkul, P. , Composing semantic web services under constraints‖, Expert Systems with Applications Volume 36, Issue 8, , 2009, pages:11021–11029.

Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y., Web Services Choreography Description Language Version 1.0. W3C Working Draft 17 December 2004, World Wide Web Consortium, 2004. Avalaible at: http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/

Keller, U. et al., Automatic location of services. In Asuncion Gomez-Pérez and Jérôme Euzenat, editors, ESWC, volume 3532 of Lecture Notes in Computer Science, pages 1–16, 2005.

Kreger, H., Web Services Conceptual Architecture (WSCA1.0), 2007. Available at: www.ibm.com/software/solutions/webservices/pdf/wsca.pdf

Lazovik, A., Aiello, M., Gennari, R., Encoding requests to web service compositions as constraints‖. Constraint programming (CP), LNCS 3709, 2005, pages.782–786.

Lécué, F., da Silva, E. M. G. et Pires, L. F., A framework for dynamic web services composition. In 2nd ECOWS Workshop on Emerging Web Services Technology (WEWST07), Halle, Germany, Germany, November 2007.

Lécué, F., Léger, A. et Delteil, A., DL Reasoning and AI Planning for Web Service Composition. In Web Intelligence, pages 445–453, 2008.

Li, J., Chen, S., Yongjun Li, Zhang, Q., Semantic Web Service Automatic Composition Based on Service Parameter Relationship Graph, trustcom, 2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, 2011, pages:1773-1778

Majithia, S., Walker, D. W. et Gray., W. A., A framework for automated service compositioninservice-orientedarchitectures. ESWS, volume 3053 of Lecture Notes in Computer Science, pages: 269–283, 2004.

Manna, Z. et Waldinger, R. J., A deductive approach to program synthesis. ACM Transactions on Programming Languages and Systems, 2(1), pages :90–121, 1980.

Martin, D. et al., OWL-S: Semantic Markup for Web Services. W3C Submission, 2004a. Available at: http://www.w3.org/Submission/OWL-S/.

Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., and Sicara, K., OWL-S : Semantic markup for web services, Tech. rep., France Telecom, MINDL Maryland, NIST, Nokia, 2004b.

McIlraith, S. A. et Son., T. C., Adapting golog for composition of semantic web services. In D. Fensel, F. Giunchiglia, D. L. McGuinness, and M.-A. Williams, editors, KR, pages: 482–496, 2002

McIlraith, S. A., Son., T. C., et Zeng., H., Semantic Web services. IEEE Intelligent Systems, 16(2), pages:46–53, March/April 2001.

Medina, J., Harbour M., et Drake J., The UML Profile for Schedulability, Performance and Time in the schedulability analysis and modeling of real-time distributed systems, Proc. of SIVOES-SPT Workshop, Toronto - Canada, May 2004

Medjahed, B., Benatallah, B., Bouguettaya, A., et Elmagarmid., A. K., Webbis : An infrastructure for agile integration of web services. Int. J. Cooperative Inf. Syst., 13(2), 2004, pages:121–158

Medjahed, B., Bouguettaya, A., A multilevel composability model for semantic Web services, Knowledge and Data Engineering, IEEE Transactions on , vol.17, no.7, pages: 954-968, July 2005

Microsoft, Installing BizTalk Server 2013. Available at: http://msdn.microsoft.com/en-us/library/jj248681(v=bts.80).aspx

Mrissa, M., Ghedira, C., Benslimane, D., et Maamar Z., A context model for semantic mediation in web services composition. In D. W. Embley, A. Oliv´e, and S. Ram, editors, ER, volume 4215 of Lecture Notes in Computer Science, Springer, 2006a, pages 12–25

Mrissa, M., Ghedira, C., Benslimane, D., et Maamar Z., Towards Context-based Mediation for Semantic Web Services Composition. In Proceedings of the Eighteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'2006), San Francisco, California, July, 2006b.

Nadalin, A., Goodner, M., Gudgin, M., Barbir, A., and Granqvist, H., WS-SecurityPolicy 1.3. OASIS standard, 2009. Available at: http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/ws-securitypolicy.pdf

Nau, D. S., Au, T. C., Ilghami, O., Kuter, U., Murdock, W., Wu, D., Yaman, F.: Shop2: An HTN Planning System. J. Artif. Intell. Res. (JAIR), Vol. 20, 2003, pages: 379–404.

Norton, B., Experiences with OWL-S, Directions for Service Composition: The Cashew Position . In OWL: Experiences and Directions Workshop (co-located with ESWC), 2005, Available at: http://www.mindswap.org/OWLWorkshop/sub23.pdf.

Omrana, H., Belouadha, F.Z. and Roudiès, O., A composability Model for efficient Web services's connectivity. IEEE International Conference of Intelligent Networking and Collaborative Systems (INCoS-2012) Romania, September 2012c, pages: 483-484

Omrana, H., Belouadha, F.Z. and Roudiès, O., Méta-modélisation des propriétés des services Web alignée avec les standards W3C, JDTIC, Tanger, 2011a.

Omrana, H., Belouadha, F.Z. et Roudiès, O., A MDA approach for describing Web services policies. The Third International Conference on Web and Information Technologies ICWIT 2010 (Marrakech), 2010b, pages: 449-460

Omrana, H., Belouadha, F.Z. et Roudiès, O., MARTE Profile-Based MDA Approach for Semantic NFP-Aware Web Services. European Journal of Scientific Research, Volume 94, issue 4, 2013a

Omrana, H., Belouadha, F.Z. et Roudiès, O., Template–based matching algorithm for dynamic web services discovery, International Journal of Information and Communication Technology, Volume 4, Issue 2/3/4, Inderscience Publishers, 2012a, pages: 198-209

Omrana, H., Belouadha, F.Z. et Roudiès, O., UML Based Profiles for Policy-aware Web services. The International Journal of Reasoning-based Intelligent Systems (IJRIS), Volume 3, Issue ¾, Inderscience Publishers, 2011b, pages: 217-225

Omrana, H., El Bitar, I., Belouadha, F.Z. and Roudiès, O., A Comparative Evaluation of Web Services Description Approaches. ITNG '13 Proceedings of the IEEE 10th International Conference on Information Technology: New Generations, Las Vegas, Nevada, USA, April 2013b, pages: 60-64

Omrana, H., Belouadha, F.Z. et Roudiès, O., A MULTI-ASPECT RULE BASED MODEL FOR WEB SERVICES OFFLINE COMPOSABILITY. Journal of Theoretical and Applied Information Technology, 2013c, (Accepted Paper).

Oppong, E., Khaddaj, S., Provision of QoS for Grid Enabled Service Oriented Architectures. Distributed Computing and Applications to Business Engineering and Science (DCABES), 2010 Ninth International Symposium on , pp.118,123, 2010.

O'Sullivan, J., Edmond., D. et Hofstede., A., Formal description of non functional service properties. Technical Report FIT-TR-2005-01, Centre for Information Technology Innovation, Queensland University of Technology, 2005

Oundhakar, S., Verma. K., Sivashanmugam, K., Sheth A. et Miller, J., Discovery of Web Services in a Federated Registry Environment, International Journal of Web Services Research, 1 (3), 2005

Pathak, J., Basu, S. and Honavar, V. Modeling Web Services by Iterative Reformulation of Functional and Non-Functional Requirements. In 4th International Conference on Service Oriented Computing, pages 314-326. LNCS 4294, Springer-Verlag, 2006.

Pereira, P.R., Service level agreement enforcement for differentiated services. In Lecture Notes in Computer Science 3883: Proc. Second International Workshop of the EURO-NGI Network of Excellence, Italy, May 2006, pages: 158–169

Phan, M. et Hattori., F., Automatic web service composition using congolog. In ICDCS Workshops, page 17, 2006.

Ponnekanti, S.R., Fox, A., Sword: A developer toolkit for web service composition, 2001

Prud'hommeaux., E., et Seaborne., A., SPARQL Query Language for RDF. W3C Recommendation, January, 2008. Avalibale at: http://www.w3.org/TR/rdf-sparql-query/

R.Waldinger. Web agents cooperating deductively. InProceedings of FAABS 2000, Greenbelt, MD, USA, April 5–7, 2000, volume 1871 of Lecture Notes in Computer Science, pages 250–262. Springer-Verlag, 2001.

Rao, J., Kungas, P. et Matskin, M., Application of Linear Logic to Web service composition. In Proceedings of the 1st International Conference on Web Services, Las Vegas, USA, June 2003.

Rao, J., Kungas, P. et Matskin, M., Logic-based Web services composition: from service description to process model. In Proceedings of the 2004 International Conference on Web Services, San Diego, USA, July 2004.

Ren, K., Chen, J., Xiao, N., Zhang, W. and Song, J., "A QSQLbased Collaboration Framework to Support Automatic Service Composition and Workflow Execution. Proceedings of the 3rd International Conference on Grid and Pervasive Computing - Workshops, 2008.

Resnik, P., Semantic Similarity in a Taxonomy: An Information-Based Measure and its Application to Problems of Ambiguity in Natural Language, Journal of Articial Intelligence Research 11, pages: 95-130, 1999.

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/

Rong, W., Liu, K.et Liang, L., Association Rule based Context Modeling for Web Service Discovery. Proceedings of the 10th IEEE Conference on E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, E-Commerce and E-Services, pages: 229-304, 2008.

Shen, L., Li, F., Ren, S., Mu, Y., Dynamic composition of web service based on coordination model. Advances in Web and Network Technologies, and Information Management, 2007, pages: 317-320

Singhera, Z.U., Extended Web services framework to meet non-functional requirements. Proceedings Symposium on Applications and the Internet - Workshops (SAINT-W'04), 2004, pages: 334-340.

Sirin, E., Parsia, B., Wu, D., Hendler, J. et Nau, D., HTN planning for web service composition using shop2. Journal of Web Semantics 1(4), pages:377- 396, 2004

Sohrabi, S., McIlraith, S.A., Optimizing web service composition while enforcing regulations. In ISWC 2009: Proceedings of the 8th International Semantic Web Conference, Chantilly, VA, USA, pages: 601–617, 2009a

Sohrabi, S., Prokoshyna, N. et McIlraith, S. A., Web service composition via the customization of golog programs with user preferences. Conceptual Modeling: Foundations and Applications, volume 5600 of Lecture Notes in Computer Science, pages: 319–334, 2009b.

Song, H., Cheng, D., Messer, A. and Kalasapur, S., Web Service Discovery Using General-Purpose Search Engines. Proceedings of the 2007 IEEE International Conference on Web Services (ICWS 2007), 2007.

Spanoudakis, G., Mahbub, K. and Zisman, A. (2007). "A Platform for Context Aware Runtime Web Service Discovery." Proceedings of the IEEE International Conference on Web Services (ICWS 2007), IEEE 2007.

Toma, I., Fensel, D., Moran, M., Iqbal, K., Strang,, T. et Roman, D., An Evaluation of Discovery approaches in Grid and Web services Environments. In The 2nd International Conference on Grid Service Engineering and Management, Erfurt, Germany, September 2005.

Vedamuthu, A., D. Orchard, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez and Ü. Yalçinalp, Web Services Policy 1.5 – Framework, W3C recommendation, 2007a. Available at: http://www.w3.org/TR/ws-policy/

Vedamuthu, A., D. Orchard, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez and Ü. Yalçinalp, Web Services Policy 1.5 – Attachment, W3C recommendation, 2007b. Available at: http://www.w3.org/TR/ws-policy-attach/

Vitvar T., Kopeck´y J., and Fensel D.. WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web. CMS WGWorking Draft, February 2009. Available at: http://cms-wg.sti2.org/TR/d11

Wang, Y. et Stroulia E., Flexible Interface Matching for WebService Discovery. Proceedings of the 4th International Conference on Web Information Systems Engineering (WISE'03), 2003

Waqar, S. et Racca, F., Business services orchestration: The hypertier of information technology, Cambridge University Press, 2004

Wohed, P., Aalst, W. M. P. v. d., Dumas, M. et Hofstede, A. H. M. t. In Lecture Notes in Computer Science, Vol. 2813/2003 (Ed, Heidelberg, S. B.). 2003

Ying, L., A Method of Automatic Web Services Composition Based on Directed Graph, cmc, vol. 1, 2010 International Conference on Communications and Mobile Computing, 2010, pages: 527-531

Zeng, L., Ngu, A. H. H. , Benatallah, B., Podorozhny, R. M., et Lei., H., Dynamic composition and optimization of web services. Distributed and Parallel Databases, 24(1-3), 2008, pages: 45–72

Zhang, P. et Li, J. (2005). "Ontology Assisted Web Services Discovery. Proceedings of the 2005 IEEE International Workshop on ServiceOriented System Engineering (SOSE'05), 2005.

Zhou, B. et Huang, T., Semantic WEB Service Discovery Search with Ontology Learning. Proceedings of the 2008 International Conference on Computer Science and Software Engineering, pages: 10481051, 2008.