HAL
open science

Sélection et composition de services Web pour la génération d'applications adaptées au contexte d'utilisation

Céline Lopez-Velasco

To cite this version:

Céline Lopez-Velasco. Sélection et composition de services Web pour la génération d'applications adaptées au contexte d'utilisation. Informatique [cs]. Université Joseph-Fourier - Grenoble I, 2008. Français. NNT : . tel-00388991

HAL Id: tel-00388991

https://theses.hal.science/tel-00388991

Submitted on 27 May 2009

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.

N° attribué par la bibliothèque
|__|__|__|__|__|__|__|__|__|__|

THÈSE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE JOSEPH FOURIER

Spécialité : Informatique

préparée au Laboratoire d'Informatique de Grenoble

dans le cadre de l'École Doctorale
Mathématiques, Sciences et Technologie de l'Information, Informatique

présentée et soutenue publiquement

par

Céline LOPEZ-VELASCO

le 18 novembre 2008

Sélection et composition de services Web pour la génération d'applications adaptées au contexte d'utilisation

---

Directeur de Thèse : Hervé MARTIN

JURY

Mme Marie-Christine FAUVET	Président
M. Djamal BENSLIMANE	Rapporteur
Mme Elisabeth MURISASCO	Rapporteur
M. Rafael LOZANO ESPINOSA	Examinateur
M. Hervé MARTIN	Directeur de Thèse
M. Jérôme GENSEL	Co-encadrant

La réalisation d'une thèse ne dépend pas d'une seule personne. De ce fait, je tiens à remercier :

Madame Marie-Christine Fauvet, Professeur à l'Université Joseph Fourier de Grenoble, qui a accepté de présider le jury et d'examiner mon travail.

Monsieur Djamal Benslimane, Professeur à l'Université Claude Bernard de Lyon, et Madame Elisabeth Murisasco, Professeur à l'Université du Sud Toulon-Var, qui m'ont fait l'honneur de rapporter mon travail et pour leurs remarques enrichissantes.

Monsieur Rafael Lozano-Espinosa, Professeur à l'Instituto Tecnológico de Estudios Superiores de Monterrey Campus Ciudad de México, qui a accepté de faire partie de mon jury en tant qu'examinateur.

Monsieur Hervé Martin, Professeur à l'Université Joseph Fourier de Grenoble, mon directeur de thèse, et Monsieur Jérôme Gensel, Professeur à l'Université Pierre Mendès France de Grenoble, mon co-encadrant. Merci à vous deux de m'avoir accueilli et conseillé afin d'assouvir au mieux mon désir de faire de la recherche, tout d'abord pendant mes stages de maîtrise et de master de recherche et ensuite durant ces années de thèse.

Ensuite je pense aux membres du Laboratoire d'Informatique de Grenoble, plus particulièrement à ceux du bâtiment D, qui m'ont croisé toutes ces années, et ceux du Laboratoire d'Informatique Fondamentale de Lille, qui commencent à me voir dans les couloirs, notamment :

Les membres de l'équipe SIGMA et STEAMER, permanents, doctorants, stagiaires CNAM, et stagiaires, que j'ai côtoyé tout au long de mon séjour au laboratoire. Je tiens à remercier tout d'abord Marlène, mon encadrante non officielle, pour sa disponibilité. Ensuite, je pense plus particulièrement à certains de mes collègues, dorénavant amis : Angela et Tati (toujours proches malgré l'Atlantique qui nous sépare), Aurélie et sa petite famille (coucou poulette !), Bogdan et Windson (ainsi que vos femmes, hein !), David et Nicolas (une pastèque bulgare, une !), Manuele (une petite bière ?), et Marius (bien plus qu'un collègue…).

Les membres de l'équipe COCOA (Areski, Bernard, Christophe, Gilles, Jean-Marc, Manu, Olivier, Raphaël, Xavier) : merci pour votre accueil et votre gentillesse.

La bande joyeuse avec qui les soirées m'ont fait oublier les difficultés rencontrées pendant la thèse : le cahors (Yan), les croque-monsieurs (Vincent), le gobage de flamby (Nicolas et Cathy – désolé Cathy !), la narcolepsie (Olivier), et le poker (Fred et Audrey).

Les membres administratifs du LIG qui m'ont toujours rendu service : Carol, Christiane, Gilles, et Pascale.

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

L'équipe pédagogique du département Information-Communication de l'IUT2 de Grenoble, pour ces trois belles années de monitorat passées en votre compagnie. Je pense particulièrement à mon tuteur, Patrick.

L'équipe pédagogique du département Techniques de Commercialisation de l'IUT2 de Grenoble, pour ma première année d'ATER.

L'équipe pédagogique du département Génie Informatique et Statistiques de Polytech'Lille, pour m'avoir sélectionné pour ma seconde année d'ATER.

Mes remerciements vont aussi …

… à mes amis :

Magali, amie d'enfance et de toujours, merci de tous ces moment partagés pendant que tu étais encore à Grenoble et de tous ceux que l'on partage, plus rarement maintenant, mais si importants, avec ta petite famille.

Les copines de Fac : Alexandra (l'européenne), Emilie (26 forever !), Louise (la globe-trotter), Mathilde et Elodie (les inséparables).

Les lames savantes (Axelle, Françoise, et Hélène) : entre ragôts, bouffe, musique, grands événements, concerts, et blagues, je n'ai partagé que des bons moments avec vous durant toutes ces années (et oui 20 ans) ! Vous m'avez vu grandir !

Les amis roumains avec qui j'ai partagé tant de bons moments. Grâce à vous tous, j'ai ainsi pu apprendre à aimer votre culture, votre musique, vos danses, votre langue, votre gastronomie, et votre sympathie !

… à ma famille :

Ma belle-famille, qui, à 2000 km en Roumanie, pense à chaque instant à nous. Vous nous manquez !

Ma sœur, mon beauf, et ma nièce : merci pour tous les moments que nous avons partagés et tous ceux que nous partagerons.

Mes parents, qui malgré toutes les difficultés de la vie, sont, je suis sûre, fiers de moi. Je n'oublirai jamais que c'est grâce à vous que je suis arrivée jusqu'ici.

Et enfin, je pense à ma toute petite famille : à mon mari, Marius, sans son amour, son aide, et son soutien, je n'aurais jamais pu mener à bien cette thèse ; et à notre petite graine, qui a partagé pendant quatre mois, bien au chaud, cette drôle d'aventure qu'est la thèse.

Tables des matières

TABLES DES MATIÈRES I

TABLES DES ILLUSTRATIONS V

1 Introduction 1
1.1 Contexte du travail 1
1.2 Problématique 3
1.2.1 Architecture orientée services 3
1.2.2 Adaptation au contexte d'utilisation 5
1.3 Aperçu de la proposition 7
1.4 Organisation du document 8

ÉTAT DE L'ART 11

2 Description de services Web 13
2.1 WSDL : le standard de description de services Web classiques 14
2.1.1 Concepts du langage WSDL 14
2.1.2 Structure du langage 15
2.1.2.1 WSDL 1.1 16
2.1.2.2 WSDL 2.0 18
2.1.2.3 Différences entre WSDL 1.1 et WSDL 2.0 21
2.2 Description de services Web sémantiques 22
2.2.1 Définition des services Web sémantiques 22
2.2.2 Langages de description de services Web sémantiques 24
2.2.2.1 OWL-S 24
2.2.2.2 SAWSDL – Annotations sémantiques pour WSDL 29
2.3 Conclusion 31

3 Publication, recherche et sélection de services Web 33
3.1 UDDI : vers des spécifications standard de publication et recherche de services Web 33
3.1.1 Publication de services Web avec UDDI 34
3.1.2 Recherche de services Web avec UDDI 35
3.2 Registres de services Web classiques accessibles sur le Web 36
3.2.1 XMethods 36
3.2.2 RemoteMethods 37
3.2.3 Vers un registre centralisé de services Web classiques : QWS Dataset 39
3.3 Solutions de publication, de recherche et de sélection de services Web sémantiques 41
3.3.1 Découverte semi-automatique de services Web centrés utilisateur 41
3.3.2 ASSAM : outil d'annotation sémantique de services Web 43
3.3.3 Approche basée sur une logique de description 44
3.3.4 Méthode de résolution de problèmes pour la sélection de services Web 45
3.4 Conclusion 46

4 Composition de services Web 47
4.1 Définitions et types de composition de services Web 47
4.1.1 Définitions 48
4.1.2 Types de composition de services Web 49
4.1.2.1 Orchestration 49
4.1.2.2 Chorégraphie 50
4.2 Langages de composition de services Web 52
4.2.1 BPEL4WS 52
4.2.2 WS-CDL 54
4.2.3 OWL-S 57
4.3 Plates-formes de composition de services Web 59

4.3.1 SELF-SERV 60
4.3.2 METEOR-S 62
4.3.3 SHOP2 64
4.3.4 IRS-III 65
4.4 Conclusion 67

5 Adaptation au contexte de services Web 69
5.1 Adaptation au contexte 70
5.1.1 Définitions du contexte 70
5.1.2 Objectifs de l'adaptation au contexte 71
5.2 Approches d'adaptation au contexte pour les services Web élémentaires 73
5.2.1 Sélection semi-automatique de services Web pour l'adaptation à l'utilisateur 73
5.2.2 Adaptation de l'exécution des services 74
5.2.3 Conversion du résultat d'un service Web mobile pour l'adaptation de la présentation du résultat 75
5.2.4 Mise en œuvre de l'adaptation dans le cadre d'une recherche de services sensibles au contexte .. 77
5.3 Approches d'adaptation au contexte pour la composition de services Web 79
5.3.1 Annotation de WSDL et médiation pour la composition adaptée au contexte des services Web . 79
5.3.2 Composition adaptée contexte à l'aide de réseaux de tâches hiérarchiques 81
5.3.3 Extension de OWL-S 83
5.3.3.1 Approche par automates d'états finis 83
5.3.3.2 Approche par planification 84
5.4 Conclusion 86

6 Synthèse de l'état de l'art 89
6.1 Représentation de services Web 89
6.2 Solutions spécifiques à la publication et à la recherche 91
6.3 Mise en œuvre de la composition de services Web 93
6.4 Mise en œuvre de l'adaptation dans le domaine des services Web 94
6.4.1 Représentation enrichie des services Web et des requêtes 95
6.4.2 Approches et techniques d'adaptation 97
6.4.3 Évaluation des techniques d'adaptation 98
6.5 Objectifs de notre proposition 100

PROPOSITION 103

7 Modèle de représentation de services Web 105
7.1 Identification des besoins en termes de représentation de services 106
7.2 WSR-Model du point de vue du fournisseur 107
7.2.1 Vue d'ensemble du modèle de représentation de services 107
7.2.2 Représentation du service 109
7.2.3 Représentation du domaine d'application 109
7.2.4 Représentation du profil fonctionnel 110
7.2.5 Représentation du profil non fonctionnel 111
7.2.6 Représentation du contexte d'utilisation 112
7.2.7 Relations entre les packages 114
7.3 WSR-Model du point de vue du client 116
7.3.1 Représentation de la requête du client 116
7.3.2 Relations entre les packages 117
7.4 Vers l'adaptation au contexte d'utilisation 118
7.5 Conclusion 120

8 Modèle de composition de services Web par méthode de résolution de problèmes 123
8.1 Définition du cycle de vie d'une composition de services Web 124
8.2 Identification des besoins en termes de composition de services 125
8.3 Outil formel pour définir la composition 126
8.3.1 Définition des méthodes de résolution de problèmes par modèle de tâches 127
8.3.1.1 Méthode de résolution de problèmes 127
8.3.1.2 Méthode de résolution de problèmes par modèle de tâches 128
8.3.2 Argumentaire sur le choix de l'approche 130

8.4 Structure du modèle de composition ProbCWS 131
8.4.1 Architecture du modèle de composition ProbCWS 132
8.4.2 Cas d'utilisation du modèle de composition ProbCWS 132
8.5 Définition des concepts de base de la composition de services Web avec ProbCWS 134
8.5.1 Définition de la tâche dans ProbCWS 134
8.5.1.1 Signature d'une tâche 135
8.5.1.2 Types de tâche 136
8.5.2 Définition de la tâche principale dans ProbCWS 137
8.5.3 Définition de la résolution de problèmes dans ProbCWS 137
8.5.3.1 La tâche élémentaire 138
8.5.3.2 La stratégie 139
8.5.4 Expression d'une résolution de problèmes 140
8.5.5 Illustration de l'utilisation du modèle de composition de services Web ProbCWS 141
8.5.5.1 Vue d'ensemble 141
8.5.5.2 Problème à résoudre 142
8.5.5.3 Sous-problème de conversion du paramètre d'entrée 143
8.5.5.4 Sous-problème de résolution 144
8.5.5.5 Sous-problème de conversion du paramètre de sortie 144
8.6 Arbre de résolution ProbCWS 145
8.6.1 Représentation des tâches 145
8.6.2 Représentation de la résolution de problèmes 146
8.6.3 Illustration d'une représentation d'une composition 147
8.7 Vers l'adaptation au contexte d'utilisation dans ProbCWS 148
8.7.1 Mise en œuvre de l'adaptation au contexte d'utilisation 148
8.7.1.1 Technique d'adaptation par déploiement de services d'adaptation 148
8.7.1.2 Technique d'adaptation par compensation 149
8.7.1.3 Technique d'adaptation par sélection de services Web 150
8.7.1.4 Technique d'adaptation par sélection de compositions 151
8.7.2 Représentation de la mise en œuvre de l'adaptation au contexte d'utilisation dans le modèle ProbCWS 152
8.7.3 Évaluation de la mise en œuvre de l'adaptation avec ProbCWS 153
8.8 Conclusion 154

9 Opérationnalisation des modèles 157
9.1 Opérationnalisation de WSR-Model 157
9.1.1 Utilisation de WSR-Model 157
9.1.2 Choix de l'outil d'opérationnalisation 158
9.1.2.1 Arguments concernant le choix d'AROM 158
9.1.2.2 Principes de base d'AROM 159
9.1.3 Mises en correspondance nécessaires à l'opérationnalisation de WSR-Model avec AROM 160
9.2 Opérationnalisation de ProbCWS 162
9.2.1 Utilisation de ProbCWS 162
9.2.2 Choix de l'outil d'opérationnalisation 163
9.2.3 Mises en correspondance des concepts de ProbCWS et ceux d'AROMTasks 164
9.2.4 Proposition d'une extension d'AROMTasks : AROMTasks-CWS 165
9.2.4.1 Mécanismes d'enrichissement de la définition d'un problème et d'une méthode 165
9.2.4.2 Mécanismes d'invocation 166
9.3 Conclusion 168

10 WSR : proposition d'un registre de services Web 169
10.1 Modélisation du registre 169
10.1.1 Objectifs 170
10.1.2 Cas d'utilisation du registre de services 170
10.1.3 Fonctionnalités du registre 171
10.1.3.1 Publication 171
10.1.3.2 Recherche 172
10.2 Implémentation du registre 173
10.2.1 Architecture 173
10.2.2 Choix technologiques 174
10.2.2.1 Noyau fonctionnel 174

10.2.2.2 Interface 174
10.2.3 Méthodes spécifiques à la publication de services Web : l'analyse d'une description WSDL 174
10.3 Utilisation de WSR 177
10.3.1 Application Web 177
10.3.1.1 Publication 178
10.3.1.2 Recherche 179
10.3.2 Services Web d'interface 181
10.3.2.1 Service Web d'interrogation de la base de connaissances 182
10.3.2.2 Service Web de publication 182
10.3.2.3 Service Web de recherche 183
10.4 Évaluation de WSR 184
10.5 Conclusion 184

11 GenAWS : une plate-forme de génération d'applications adaptées au contexte d'utilisation à base de services Web 187
11.1 Modélisation de la plate-forme GenAWS 188
11.1.1 Objectifs 188
11.1.2 Cas d'utilisation de la plate-forme GenAWS 188
11.1.3 Fonctionnalités de la plate-forme 190
11.2 Implémentation de la plate-forme 192
11.2.1 Architecture 192
11.2.2 Choix technologiques 193
11.2.2.1 Noyau fonctionnel 193
11.2.2.2 Interface 194
11.3 Utilisation de la plate-forme GenAWS 194
11.3.1 Application Web 194
11.3.1.1 Définition de la composition 195
11.3.1.2 Génération d'applications 196
11.3.2 Services Web d'interface 197
11.3.2.1 Service Web de définition d'une composition 197
11.3.2.2 Service Web de génération d'applications 198
11.4 Évaluation de GenAWS 198
11.5 Conclusion 199

CONCLUSION 201

12 Bilan et perspectives 203
12.1 Bilan du travail réalisé 203
12.2 Perspectives 205

BIBLIOGRAPHIE 207

ANNEXES 223
Annexe 1 : OntoWS – instanciation du modèle de représentation de services dans le contexte du Web sémantique 223
Ontologie du service 224
Ontologie du domaine d'application 226
Ontologie du contexte d'utilisation 227
Ontologie OntoWS 229
Annexe 2 : Signatures des méthodes des services Web d'interface de WSR 231
Annexe 3 : Signatures des méthodes des services Web d'interface de GenAWS 232

Tables des illustrations

Figures

Figure 1.1 Illustration du cycle de vie d'une application à base d'architecture orientée services. 4
Figure 1.2 Correspondance des chapitres composant ce mémoire avec les étapes du cycle de vie d'une application à base d'architecture orientée services. 8
Figure 2.1 Structure générale d'un document WSDL 1.1. 16
Figure 2.2 Illustration de l'utilisation de l'élément `definitions` dans la description du service Web *GlobalWeather* en WSDL 1.1. 16
Figure 2.3 Illustration de l'utilisation de l'élément `types` dans la description du service Web *GlobalWeather* en WSDL 1.1. 17
Figure 2.4 Illustration de l'utilisation de l'élément `message` dans la description du service Web *GlobalWeather* en WSDL 1.1. 17
Figure 2.5 Illustration de l'utilisation des éléments `portType` et `operation` dans la description du service Web *GlobalWeather* en WSDL 1.1. 17
Figure 2.6 Illustration de l'utilisation de l'élément `binding` dans la description du service Web *GlobalWeather* en WSDL 1.1. 18
Figure 2.7 Illustration de l'utilisation de l'élément `service` dans la description du service Web *GlobalWeather* en WSDL 1.1. 18
Figure 2.8 Structure générale d'un document WSDL 2.0. 19
Figure 2.9 Illustration de l'utilisation de l'élément `description` dans la description du service Web *GlobalWeather* en WSDL 2.0. 19
Figure 2.10 Illustration de l'utilisation de l'élément `types` dans la description du service Web *GlobalWeather* en WSDL 2.0. 19
Figure 2.11 Illustration de l'utilisation des éléments `interface` et `operation` dans la description du service Web *GlobalWeather* en WSDL 2.0. 20
Figure 2.12 Illustration de l'utilisation de l'élément `binding` dans la description du service Web *GlobalWeather* en WSDL 2.0. 20
Figure 2.13 Illustration de l'utilisation de l'élément `service` dans la description du service Web *GlobalWeather* en WSDL 2.0. 21
Figure 2.14 Les services Web sémantiques se situent entre deux problématiques liées au Web : l'interopérabilité et l'annotation sémantique des ressources, d'après [Fensel *et al.,* 2002a]. 23
Figure 2.15 Diagramme fonctionnel de l'ontologie de service, d'après [Martin *et al.*, 2004]. 24
Figure 2.16 Ontologie *Service* représentant l'ensemble des éléments composant la description sémantique du service Web *GlobalWeather* par le biais de OWL-S. 25
Figure 2.17 *ServiceProfile* du service Web *GlobalWeather* défini par l'élément `Profile`. 26
Figure 2.18 Extrait de l'ontologie *Concept* pour la description des types de données du service Web *GlobalWeather*. 26
Figure 2.19 *ServiceModel* du service Web *GlobalWeather*. 27
Figure 2.20 *ServiceGrounding* du service Web *GlobalWeather*. 28
Figure 2.21 Extrait de l'ontologie *weather-ont*, illustrant la classe `WeatherObservation`. 30
Figure 2.22 Annotation sémantique de l'élément `type` de la description WSDL 2.0 du service *GlobalWeather*. 30
Figure 3.1 Entités composant un annuaire UDDI, d'après [Keidl *et al.,* 2004]. 34
Figure 3.2 Extrait de la liste des services Web disponibles sur *XMethods*. 37
Figure 3.3 Catégories de services pour la recherche de services sur le site *RemoteMethods*. 38
Figure 3.4 Interface de recherche simple de *QWS Dataset*. 39
Figure 3.5 Résultat, dans *QWS Dataset*, de la recherche définie précédemment (*cf.* Figure 3.4) contenant les informations sur les services Web correspondant à la requête, utiles au processus de sélection. 40

Figure 4.1 Illustration du cycle de vie de d'une composition de services Web par [Benatallah *et al.*, 2002]. 48
Figure 4.2 Illustration de l'orchestration, d'après [Peltz, 2003]. 50
Figure 4.3 Vue générale de l'orchestration. 50
Figure 4.4 L'illustration de la chorégraphie, d'après [Peltz, 2003]. 51
Figure 4.5 Vue générale de l'exécution d'une composition de services Web de type chorégraphie. 51
Figure 4.6 Le flot de processus avec BPEL4WS, d'après [Peltz, 2003]. 53
Figure 4.7 Définition du processus exécutable de la demande du service de météorologie *myMeteo* à l'aide de BPEL4WS. 54
Figure 4.8 Représentation du package de WS-CDL, d'après [Barros *et al.,* 2005a]. 55
Figure 4.9 Définition du rôle et du comportement du service *GeoIPService*. 55
Figure 4.10 Définition de la relation entre les services Web *GeoIPService* et *GlobalWeather*. 56
Figure 4.11 Extrait de l'interaction entre les deux services Web *GeoIPService* et *GlobalWeather* pour implémenter la composition *myMeteo*. 57
Figure 4.12 Description des différents types de processus, d'après [Martin *et al.*, 2004]. 58
Figure 4.13 Extrait du modèle de processus (*ProcessModel*) de la composition *myMeteo*. 59
Figure 4.14 Architecture de SELF-SERV, d'après [Sheng *et al*., 2002]. 60
Figure 4.15 Illustration du fonctionnement de SELF-SERV à l'aide de l'exemple de la composition *MyMeteo*. 61
Figure 4.16 Architecture de METEOR-S, d'après [Aggarwal *et al.*, 2004]. 62
Figure 4.17 Illustration du fonctionnement de METEOR-S à l'aide de l'exemple de la composition *MyMeteo*. 63
Figure 4.18 Architecture de SHOP2, d'après [Sirin *et al*., 2004]. 64
Figure 4.19 Illustration du fonctionnement de SHOP2 à l'aide de l'exemple de la composition *MyMeteo*. 65
Figure 4.20 Architecture de IRS-III [Cabral *et al.,* 2006]. 66
Figure 4.21 Illustration du fonctionnement d'IRS-III à l'aide de l'exemple de la composition *MyMeteo*. 67
Figure 5.1 illustration d'une requête de type SQL exprimée dans [Balke *et al.,* 2003b]. 73
Figure 5.2 Composants pour l'adaptation des services Web par contexte, d'après [Keidl *et al.,* 2004]. 74
Figure 5.3 Conversion dynamique de contenu du résultat d'un service Web [Pashtan *et al.,* 2004]. 76
Figure 5.4 Architecture de recherche de services sensibles au contexte de l'utilisateur et illustration des trois mécanismes de filtrage, d'après [Suraci *et al.*, 2007]. 77
Figure 5.5 Annotation orientée contexte d'un paramètre d'une opération d'un service Web, d'après [Mrissa, 2007] 80
Figure 5.6 Vue détaillée du service Web médiateur, d'après [Mrissa, 2007]. 80
Figure 5.7 Illustration de la définition de la requête par l'utilisateur, d'après [Vukovic *et al*., 2004]. 81
Figure 5.8 Architecture du système pour la composition dynamique de services sensibles au contexte, d'après [Vukovic *et al.,* 2004]. 82
Figure 5.9 OWL-SC, extension de OWL-S : ajout des ontologies de contexte dans la description de services Web, d'après [Qiu *et al.,* 2006]. 84
Figure 5.10 Architecture proposée par [Qiu *et al.,* 2006] pour composer des services Web adaptés au contexte d'utilisation, d'après [Qiu *et al.,* 2006]. 85
Figure 6.1 Typologie retenue pour l'état de l'art des techniques d'adaptation dans le domaine des services Web (inspirée de [Villanova-Oliver, 2002]). 99
Figure 6.2 Illustration des moyens mis en œuvre pour atteindre nos objectifs. 100
Figure 7.1 Vue d'ensemble du modèle de représentation de services Web. 107
Figure 7.2 Classes et associations du package *Service*. 109
Figure 7.3 Classes et associations du package *Application Domain*. 109
Figure 7.4 Classes et associations du package *Functional Profile*. 111
Figure 7.5 Classes et associations du package *Non Functional Profile*. 111
Figure 7.6 Description du contexte vu comme une composition d'éléments de contexte, d'après [Kirsch Pinheiro, 2006]. 113
Figure 7.7 Classes et associations du package *Use Context*. 113

Figure 7.8 Diagramme de classes représentant les relations entre les cinq packages composant le modèle de représentation de services (NB : toutes les classes et associations du modèle ne sont pas représentées sur cette figure). 115
Figure 7.9 Classes et association du package *Query Facilities*. 116
Figure 7.10 Diagramme de classes représentant les relations entre le package *Query* et les packages représentant le service (N.B. : seules les classes et associations mises en jeu pour formaliser la requête sont représentées sur cette figure). 117
Figure 7.11 Diagramme de classes rappelant les relations entre les packages *Service* et *Use Context*. 119
Figure 8.1 Diagramme d'activités UML représentant le cycle de vie d'une composition de services Web, inspiré de [Benatallah *et al.*, 2002]. 124
Figure 8.2 Architecture des applications utilisant des méthodes de résolutions de problèmes, d'après [Gómez Pérez *et al.,* 1999]. 128
Figure 8.3 Exemple d'un arbre de résolution d'une résolution de problème (issu de la tâche T) par modèle de tâches. 130
Figure 8.4 Architecture pour la mise en œuvre du modèle de composition de services Web ProbCWS. 132
Figure 8.5 Cas d'utilisation du modèle de composition de services ProbCWS. 133
Figure 8.6 Extrait du package *Task*. 134
Figure 8.7 Diagramme de classes représentant la signature d'une tâche via les relations entre la classe *Task* du package *Task* et d'autres classes issues des packages *Application Domain*, *Functional Profile*, et *Use Context*. 135
Figure 8.8 Diagramme de classes représentant la classe *MainTask* et le lien avec le package *Composition*. 137
Figure 8.9 Diagramme de classes représentant les liens entre la classe *ElementaryTask* du package *Task* et d'autres classes issues des packages *Task*, *Service*, et *Use Context*. 138
Figure 8.10 Extrait du package *Task* représentant la stratégie. 139
Figure 8.11 Instructions de la tâche élémentaire *compositionTask*. 142
Figure 8.12 Instructions de la stratégie myMeteo. 143
Figure 8.13 Instructions de la tâche élémentaire *geoIP*. 143
Figure 8.14 Instructions de la tâche élémentaire *globalWeather*. 144
Figure 8.15 Instructions de la tâche *xml2dial*. 144
Figure 8.16 Notation graphique d'une tâche principale. 145
Figure 8.17 Notation graphique des tâches élémentaires. 146
Figure 8.18 Notation graphique des instructions des tâches élémentaires. 146
Figure 8.19 Notation graphique d'une stratégie. 146
Figure 8.20 Exemple de l'illustration graphique d'une résolution de problèmes avec l'arbre de résolution ProbCWS. 146
Figure 8.21 Représentation via l'arbre de résolution ProbCWS de la résolution du problème *myMeteo*. 147
Figure 8.22 Illustration de la mise en œuvre de la technique d'adaptation par compensation à l'aide de l'arbre de résolution ProbCWS. 150
Figure 8.23 Illustration de la mise en œuvre de la technique d'adaptation par sélection de services Web via un arbre de résolution ProbCWS. 151
Figure 8.24 Illustration de la technique d'adaptation par sélection de compositions. 151
Figure 8.25 Diagramme de classes représentant le package *Adaptation* et le lien avec le package *Use Context*. 152
Figure 8.26 Critères d'évaluation de la mise en œuvre de l'adaptation avec ProbCWS. 153
Figure 9.1 Architecture simplifiée d'un registre. 158
Figure 9.2 Principes élémentaires des bases de connaissances AROM représentés, à gauche, graphiquement, et à droite, textuellement. 160
Figure 9.3 Package *Functional Profile* de WSR-Model (en haut) et son opérationnalisation en AROM (en bas). 161
Figure 9.4 Rappel de l'architecture préalable à la mise en œuvre de ProbCWS. 162
Figure 9.5 Environnement intégré d'AROM, d'après [Chabalier, 2004]. 163
Figure 9.6 Syntaxe d'un problème dans AROMTasks. 163

Figure 9.7 Syntaxe d'une méthode dans AROMTasks. 164
Figure 9.8 Syntaxe d'une stratégie dans AROMTasks. 164
Figure 9.9 Illustration de la mise en correspondance des principes de base d'AROMTasks et de ProbCWS. 164
Figure 9.10 Illustration du mécanisme d'intégration de la spécification du contexte d'utilisation et du domaine d'application aux méthodes AROMTasks-CWS. 165
Figure 9.11 Illustration de l'intégration des spécifications AROMTasks-CWS dans AROMTasks. 166
Figure 9.12 Illustration des mécanismes d'invocation dans AROMTasks-CWS. 167
Figure 9.13 Illustration de l'intégration des spécifications d'invocation dans AROMTasks. 167
Figure 10.1 Cas d'utilisation du registre de services. 171
Figure 10.2 Diagramme de séquences UML représentant la publication d'un service à l'aide de WSR. 171
Figure 10.3 Diagramme de séquences UML représentant la recherche de services à l'aide de WSR. 172
Figure 10.4 Architecture du registre de services WSR. 173
Figure 10.5 Processus d'extraction d'une description WSDL vers les instances d'une base de connaissances AROM selon le modèle de représentation de services Web. 175
Figure 10.6 Analyse de la description WSDL du service Web *GlobalWeather* lors de sa publication dans WSR. 176
Figure 10.7 Associations déduites de l'analyse de la description WSDL du service Web *GlobalWeather* lors de sa publication dans WSR. 177
Figure 10.8 Page d'accueil de l'application Web de WSR. 177
Figure 10.9 Illustration de l'utilisation de l'application Web concernant la publication d'un service Web dans WSR. 178
Figure 10.10 Illustration de l'utilisation de l'application Web concernant l'association du contexte d'utilisation au service à publier dans WSR. 179
Figure 10.11 Recherche d'un service Web élémentaire : à gauche par nom de service, à droite par catégorie de services. 180
Figure 10.12 Recherche en vue de concevoir une application adaptée au contexte d'utilisation. 180
Figure 11.1 Cas d'utilisation de la plate-forme GenAWS. 189
Figure 11.2 Illustration des transferts de données dans le cadre de la génération d'applications avec GenAWS. 190
Figure 11.3 Architecture de la plate-forme GenAWS. 192
Figure 11.4 Page d'accueil de l'application Web de GenAWS. 195
Figure 11.5 Formulaire de l'application Web de GenAWS pour l'enregistrement d'une définition de composition définie en ProbCWS. 195
Figure 11.6 Illustration de la page Web de génération d'applications au sein de l'application Web de GenAWS. 196

Tableaux

Tableau 2.1 Comparaison des éléments XML utilisés dans chacune des versions de WSDL pour décrire les concepts de description de services. 21
Tableau 6.1 Catégories d'information relatives aux services Web proposées par les travaux portant sur la publication, recherche et sélection de services Web. 90
Tableau 6.2 Publication et méthodes de recherche mises en œuvre par les travaux portant sur la publication, recherche et sélection de services Web. 92
Tableau 6.3 Synthèse des méthodes utilisées et solutions apportées par les plates-formes de composition de services Web. 93
Tableau 6.4 Catégories d'information du contexte enrichissant les représentations des services Web et de la requête du client, dans les travaux d'adaptation de services Web élémentaires ou composants. 96
Tableau 6.5 Synthèse des travaux portant sur l'adaptation des services Web en soulignant les formes et techniques d'adaptation mises en œuvre. 98
Tableau 7.1 Packages du modèle de représentation de services Web associés aux catégories d'information identifiées précédemment. 108
Tableau 8.1 Bilan des formes et des techniques d'adaptation au contexte d'utilisation mises en œuvre dans ProbCWS. 153
Tableau 9.1 Tableau récapitulant les mises en correspondance nécessaires pour l'opérationnalisation de WSR-Model à l'aide du système AROM. 160
Tableau 12.1 Tableau de synthèse de la mise en œuvre de l'adaptation au contexte d'utilisation au sein de nos propositions. 204

1 INTRODUCTION

1.1 Contexte du travail

L'Architecture Orientée Services (AOS ou SOA – *Service-Oriented Architecture* en anglais) permet aux concepteurs de systèmes d'information d'organiser un ensemble de logiciels isolés en un ensemble de services interconnectés, accessibles par une interface et des protocoles standard [Papazoglou, 2003]. D'après [Jones, 2005], un *service* est une unité discrète qui remplit une collection de tâches répondant au(x) même(s) objectif(s). L'AOS est un paradigme architectural qui peut être utilisé pour concevoir des infrastructures permettant aux *clients* et aux *fournisseurs* d'échanger des services, malgré la disparité des domaines, des technologies, et des fournisseurs [Nickul *et al*., 2005]. Dans notre travail, nous nommons *clients* les entités qui ont des besoins en termes de services, tels que les concepteurs de systèmes, et *fournisseurs* les entités qui offrent les services. Selon [Erl, 2005], un ensemble de facteurs clés fait converger les clients vers une architecture orientée services. Nous en décrivons trois qui, de notre point de vue, sont essentiels : la réutilisation, la découverte, et la composition.

La réutilisation. Les unités logiques des architectures des systèmes sont divisées en services dans l'intention de promouvoir leur réutilisation. Nous appelons ces services des *services élémentaires*. Selon [Nickul *et al*., 2005], cette réutilisation nécessite que chaque service utilisé dans un système basé sur une AOS soit, au préalable, décrit par son fournisseur. L'avantage de la réutilisation est un gain de temps et d'investissement lors de la conception d'une nouvelle application.

La découverte. Les services sont conçus afin d'être sélectionnés via des mécanismes de recherche. La découverte est permise par la description préalable des services et leur publication au sein d'un registre. La découverte est un point clé nécessaire à la réutilisation des services élémentaires dans une AOS.

La composition. Des collections de services peuvent être coordonnées et assemblées afin de former une composition de services. Cette possibilité de construire de nouveaux systèmes à partir de services existants constitue un des avantages de l'AOS.

Dans le contexte de mise en œuvre d'applications à base d'architecture orientée services, la technologie la plus utilisée est celle des services Web [Erl, 2005]. [Haas *et al.,* 2004] définissent un

service Web comme un logiciel conçu pour supporter l'interaction entre machines interopérables à travers un réseau. Cette interopérabilité est rendue possible grâce à deux standards connus par l'ensemble des acteurs (fournisseurs et clients). Le premier est le langage de description (WSDL – *Web Service Description Language*, [Chinnici *et al.*, 2007]) et le second est le protocole de communication (SOAP – *Simple Object Access Protocol*, [Mitra *et al.*, 2007]).

L'avantage principal de l'architecture orientée services en général et Web en particulier, est que ce paradigme pourrait répondre aux besoins en termes de flexibilité et d'adaptabilité rapide exprimés par les concepteurs [Crusson, 2003]. Une application à base de services est *flexible* puisqu'un service défaillant peut être remplacé par un autre sans modifier l'ensemble de l'application. De plus, elle est *adaptable* du fait que le service sélectionné est celui choisi comme étant le meilleur dans un contexte donné. Ce type d'architecture est aujourd'hui d'autant plus populaire qu'il est amplement utilisé par les organisations qui veulent prendre part au Web 2.0.

Le terme Web 2.0 a été proposé par Tim O'Reilly et Dale Dougherty en 2004 afin de nommer les nouvelles applications et émergences du Web. La définition du Web 2.0 par [O'Reilly, 2005] repose sur sept principes :

- le Web en tant que plate-forme (*the Web as platform*),
- tirer parti de l'intelligence collective (*harnessing collective intelligence*),
- la puissance est dans les données (*data is the next intel inside*),
- la fin des cycles de versions (*end of the software release cycle*),
- des modèles de programmation légers (*lightweight programming models*),
- le logiciel se libère du dispositif (*software above the level of a single device*), et
- enrichir les interfaces utilisateur (*rich user experiences*).

Nous en décrivons trois que nous jugeons pertinents dans le cadre d'un rapprochement du Web 2.0 et de l'AOS : le Web en tant que plate-forme d'exécution, tirer parti de l'intelligence collective et des modèles de programmation légers.

Le Web en tant que plate-forme. Ce principe sous-entend que le Web et les dispositifs qui lui sont connectés sont considérés comme une plate-forme globale de services réutilisables.

Tirer parti de l'intelligence collective. Les utilisateurs du Web rendent disponible à tous sur la toile leur intelligence. Par intelligence nous pensons aux blogs, aux participations aux Wikis, qui seront lus par d'autres utilisateurs, mais aussi aux services disponibles sur le Web qui peuvent être utilisés (et réutilisés) par le plus grand nombre.

Des modèles de programmation légers. Ce principe préconise aux fournisseurs de services élémentaires de déployer les services afin qu'ils soient facilement utilisables et indépendants les uns des autres. Ceci facilitera l'implémentation d'applications faiblement couplées et évolutives.

Divers types d'application émergent du concept du Web 2.0, notamment le *mashup*. Les applications *mashup* sont des applications hybrides dans lesquelles au moins deux services ou technologies sont combinés dans le but de créer un nouveau service [Maness, 2006]. Ce type d'application inclut les mêmes problématiques que les composition de services Web : le partage, la sélection, la réutilisation, et l'intégration des services [Benslimane *et al.*, 2008]. Les applications *Mobile Web 2.0* – un autre type d'application Web 2.0 – regroupent les services Web 2.0 dont l'accès et l'utilisation sont réalisés à travers des dispositifs mobiles [Wahlster *et al.*, 2006]. L'utilisation nomade de systèmes peut entraîner une prise en compte, lors de la conception de ces applications, du *contexte* afin de proposer aux utilisateurs des informations et services adaptés. Le contexte (par

exemple, la localisation de l'utilisateur, les dispositifs qu'il a à sa disposition) permet de mettre en œuvre des applications adaptées au contexte et ainsi valorise ce type de logiciel.

Le concept de Web 2.0 peut être enrichi par le domaine du Web sémantique. Certains auteurs dénomment cette combinaison le Web 3.0 [Wahlster *et al.*, 2006] [Hendler, 2008]. L'objectif premier du Web sémantique utilisé de manière isolée est de définir et de lier sémantiquement les ressources du Web en général, et les services Web en particulier, afin de simplifier leur utilisation, leur découverte, leur intégration, et leur réutilisation dans le plus grand nombre d'applications [Berners-Lee *et al.*, 2001]. Ces aspects du Web sémantique peuvent faciliter la description, la recherche et la sélection de services Web dans des applications à base d'AOS et/ou Web 2.0. Le Web 3.0 permet aux applications sémantiques (issues du domaine du Web sémantique) de tirer profit des avantages mis en avant par le Web 2.0, tels que les modèles de programmation légers ou la présence de communautés sur le Web. En effet, le coût important de la mise en place du Web sémantique peut être réduit par l'intermédiaire de modèles de programmation légers ou en exploitant les communautés Web 2.0 pré-existantes. Inversement, dans le cadre du Web 3.0, les applications issues du Web 2.0 se trouvent enrichies par l'apport sémantique fourni par le domaine du Web sémantique. Les difficultés liées à la mise en œuvre de *mashup*, telles que la recherche et la sélection de services élémentaires, peuvent être surpassées avec l'utilisation du Web sémantique.

Cette thèse a pour objectif de faciliter la tâche des concepteurs d'applications à base de services Web afin de mettre en œuvre de telles applications. Nous nous intéressons particulièrement à quatre aspects : la réutilisation des services Web élémentaires, la découverte et la sélection des services Web, et la composition des services Web sélectionnés en vue d'implémenter l'application désirée. De manière transversale, nous nous intéressons aussi à la sémantique des services Web et à la mise en œuvre de l'adaptation au contexte d'utilisation.

1.2 Problématique

1.2.1 Architecture orientée services

La mise en œuvre d'applications à base d'AOS repose sur un cycle de vie composé de trois phases (phase préliminaire, phase de modélisation, et phase d'assemblage) dans lesquelles interviennent deux types d'acteurs (un ou plusieurs fournisseur(s) et un client) [Yu *et al.*, 2005]. La Figure 1.1 illustre les différentes phases du cycle de vie d'une AOS ainsi qu'une partie des étapes constituant ces phases. Dans cette figure, nous mettons aussi en évidence les étapes du cycle de vie spécifiques aux services Web.

La première phase du cycle de vie est la **phase préliminaire** réalisée par un **fournisseur** de services élémentaires. Cette phase est composée de trois étapes : le déploiement, la description, et la publication du service Web. Nous décrivons ici plus particulièrement les deux dernières étapes qui s'inscrivent dans la problématique de notre travail. L'étape de déploiement est spécifique à chaque fournisseur et correspond à l'implémentation du service qui, par la suite, sera décrit et publié.

La description. Afin que le service Web déployé puisse être visible, dans le but d'être utilisé (et réutilisé), le fournisseur doit le décrire à l'aide du standard WSDL. Cependant, les informations fonctionnelles du service Web contenues dans une description WSDL (nom et paramètres des méthodes proposées) ne suffisent pas à elles seules à mettre en œuvre des mécanismes de sélection répondant aux besoins des futurs clients. Afin de combler ce manque en termes de représentation de services Web, des propositions issues du Web sémantique ont vu le jour (telles que OWL-S [Martin *et al.*, 2004] et SAWSDL [Farrell *et al.,* 2007]), mais aucune ne propose une représentation de service assez large pour mettre en œuvre des applications adaptées au

contexte d'utilisation via des services Web. Cependant, il est en envisageable de les étendre afin de prendre en compte les notions nécessaires à l'adaptation (par exemple, [Qiu *et al.*, 2006] intègrent une représentation de l'utilisateur à OWL-S).

La publication. La recherche des services en vue de les utiliser (et réutiliser) repose sur leur publication préalable dans un registre. UDDI (*Universal Description Discovery, and Integration*) [Clement *et al.,* 2004] avait à l'origine l'ambition de devenir le registre universel de services Web. Cependant, à ce jour, lorsqu'il est utilisé, il l'est localement au sein d'une entreprise et n'est pas accessible aux acteurs externes à cette entreprise. De nombreux travaux proposent des registres publics sur le Web (tels que les sites *XMethods*[1] ou *RemoteMethods*[2]), mais peu offrent une représentation détaillée des services Web afin d'en accroître la visibilité.

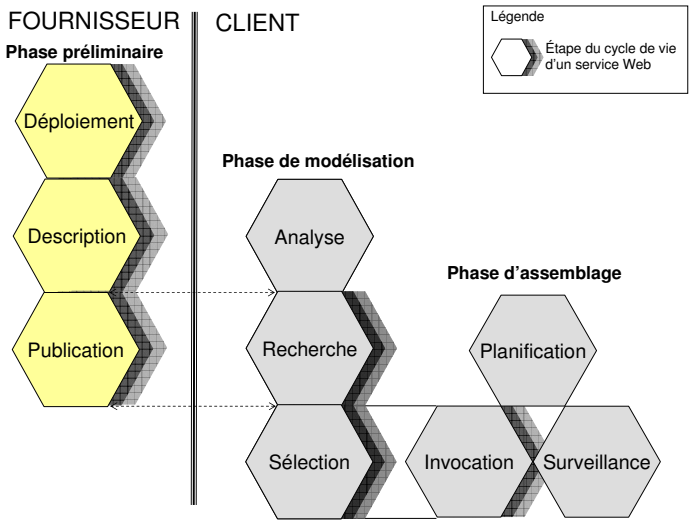


Figure 1.1 Illustration du cycle de vie d'une application à base d'architecture orientée services[3].

Les deux phases suivantes du cycle de vie sont réalisées par le **client** (le concepteur d'applications à base d'AOS). La **phase de modélisation** est composée de trois étapes. La première étape est l'analyse des besoins préalable à la conception et se situe au niveau interne des organisations. Nous nous préoccupons plus particulièrement des deux étapes suivantes : la recherche et la sélection de services Web.

La recherche. La recherche peut se faire via les registres dans lesquels les fournisseurs ont publiés leurs services Web. Cependant, les registres existants ne proposent pas de mécanismes de recherche avancée (par exemple, le travail *QWSDataset* [Al-Masri *et al.*, 2007b] n'offre qu'une recherche par mots-clés[4]). De plus, peu de services Web sont disponibles dans les registres publics. Par exemple, le site de référence de programmation Web 2.0, *ProgrammableWeb*[5], ne propose qu'environ 700 API, et peu sont des services Web (environ 180).

La sélection. Le client doit ensuite sélectionner parmi la collection de services Web issus de l'étape de recherche, le service Web qui convient le mieux à ses attentes. Cependant, les représentations existantes des services Web, soit réalisées par les langages de description, soit stockées dans les registres de publication, ne sont pas suffisamment précises pour faciliter la

[1] http://www.xmethods.com

[2] http://www.remotemethods.com

[3] Seules les étapes du cycle de vie pertinentes pour notre travail sont représentées dans cette figure.

[4] http://www.uoguelph.ca/~qmahmoud/qws/search.html

[5] http://www.programmableweb.com/

tâche du client. En effet, les descriptions issues des langages tels que WSDL ne contiennent que les aspects fonctionnels des services (nom des méthodes proposées, paramètres d'entrée et de sortie), et celles issues de la publication des services au sein de registres ne contiennent pas assez d'information (telle que les fonctionnalités remplies par les services) pour faciliter la sélection.

La dernière phase du cycle de vie est la **phase d'assemblage**. Cette phase est composée de trois étapes (la planification, l'invocation, et la surveillance). Ces étapes sont répétées jusqu'à l'implémentation intégrale de l'application.

La planification. Lors de cette étape, le client définit la composition de services Web. Il existe de nombreux langages de composition de services Web (tels que BPEL4WS [Andrews *et al.,* 2003] ou OWL-S [Martin *et al.*, 2004]). Cependant, ces définitions sont, soit rigides en termes d'exécution (comme avec l'utilisation de BPEL4WS), soit complexes à décrire (comme avec l'utilisation de OWL-S).

L'invocation. Lorsque le client connaît au préalable le service Web qu'il veut invoquer (par l'intermédiaire des étapes précédentes telles que la recherche et la sélection), son invocation peut être réalisée via le protocole de communication standard SOAP.

La surveillance. L'exécution d'une composition de services Web engendre des problèmes tels que l'indisponibilité d'un service Web. Lorsqu'un service Web faisant partie d'une composition est indisponible, l'étape de surveillance doit reposer sur des *mécanismes de compensation* afin de sélectionner un nouveau service Web répondant au mieux aux objectifs du service Web défaillant. Or, aujourd'hui, peu de plates-formes de composition de services Web proposent de tels mécanismes de compensation.

1.2.2 Adaptation au contexte d'utilisation

Selon [Dey, 2001], *le contexte est n'importe quelle information qui peut être utilisée pour caractériser la situation d'une entité. Une entité est une personne, un endroit ou un objet qui est considérée pertinente dans l'interaction entre un utilisateur et une application, incluant l'utilisateur et les applications elles-mêmes*. Dans notre travail, nous nous focalisons sur les éléments du contexte qui caractérisent l'utilisation d'un système, pouvant être détectés et utilisés par le système pour offrir un résultat adapté. Nous appelons cet ensemble d'éléments le *contexte d'utilisation*. Nous considérons quatre éléments principaux composant la description du contexte d'utilisation : *l'utilisateur* (son profil, ses droits d'accès, son activité, etc.), sa *localisation* (coordonnées GPS, lieu où l'utilisateur se situe, etc.), le *temps* (heure, jour, mois, etc.) et le *dispositif utilisé* (caractéristiques logicielles et matérielles du dispositif d'accès, le réseau utilisé, etc.).

Lors de la réalisation d'applications, les concepteurs doivent prendre en compte un ensemble de paramètres que nous dénommons le contexte d'utilisation (utilisateur, temps, localisation, dispositif) afin d'adapter le fonctionnement de l'application à ce contexte. Dans ce cas, l'utilisation d'une architecture orientée services, particulièrement services Web, permet que la mise en œuvre proprement dite de l'application soit déléguée à des entités externes, les services, choisis selon le contexte donné. Ainsi l'exécution est réalisée sur le Web, et non sur les dispositifs mobiles à faible ressource.

Dans les trois phases du cycle de vie (*cf.* Figure 1.1), il est envisageable de mettre en œuvre une ou plusieurs formes d'adaptation.

Prise en compte de l'adaptation lors de l'étape de déploiement. Nous considérons deux manières de mettre en œuvre l'adaptation au sein de l'étape de déploiement : soit par le biais de services Web adaptés, soit avec des services Web d'adaptation.

- *Service Web adapté*. Le fournisseur de services peut, lors de l'étape de déploiement, implémenter les services Web afin qu'ils répondent de manière adaptée à une situation donnée. Par exemple, un même service de météorologie peut, lorsqu'il est exécuté sur un ordinateur de bureau, renvoyer des images, et lorsqu'il est exécuté sur un dispositif mobile, seulement renvoyer les informations météorologiques au format texte. Un *service Web adapté* est donc un service Web qui exécute une tâche quelconque mais a la caractéristique de s'adapter.
- *Service Web d'adaptation*. À ce jour, à notre connaissance, aucun travail portant sur l'adaptation des services Web au contexte d'utilisation ne propose de mettre en œuvre des services Web d'adaptation. Nous appelons *services Web d'adaptation* des services qui ont pour seul objectif l'adaptation à un contexte d'utilisation particulier. Par exemple un tel service convertit le résultat d'un autre service afin qu'il soit lisible sur un dispositif mobile.

Dans les deux cas (utilisation d'un service Web adapté ou d'un service Web d'adaptation), afin que les clients de ces services sachent que les services sont adaptés à une situation particulière, il est nécessaire d'améliorer la description classique d'un service Web faite à l'aide de WSDL afin d'y intégrer le (ou les) aspect(s) du contexte d'utilisation au(x)quel(s) le service s'adapte.

Prise en compte de l'adaptation lors des étapes de recherche et de sélection. Dans la phase de modélisation, les étapes de recherche et de sélection peuvent intégrer une forme d'adaptation en filtrant dans la collection de services Web disponibles sur le(s) registre(s), ceux correspondant au mieux, tant aux besoins fonctionnels du client, qu'aux besoins en termes d'adaptation au contexte d'utilisation (profil de l'utilisateur, type de dispositif utilisé, temps et lieu de l'utilisation, etc.). Ceci implique d'associer à la requête classique du client le besoin d'adaptation. L'enrichissement de la requête ne suffit pas à mettre en œuvre l'adaptation, il faut aussi intégrer à la description du service Web le contexte d'utilisation auquel il est adapté. Or, peu de travaux portant sur l'adaptation des services Web proposent cet enrichissement de la description de service Web.

Prise en compte de l'adaptation lors des étapes de planification et de surveillance. Lors de la phase d'assemblage, deux formes d'adaptation peuvent être envisagées. La première se situe au niveau de l'étape de planification lors de laquelle le client peut indiquer un ensemble de définitions de composition de services possibles, et choisir celle qui correspond le mieux aux besoins en termes d'adaptation. La seconde forme d'adaptation se situe au niveau de l'étape de surveillance et consiste à vérifier, pour chaque niveau de la composition, que le service choisi répond à la demande d'adaptation. Ces formes d'adaptation sont mises en œuvre dans certains travaux portant sur l'adaptation de la composition de services Web, mais aucun n'offre la possibilité de combiner ces types d'adaptation.

Les travaux issus du domaine des services Web portent sur une ou plusieurs étapes du cycle de vie des applications à base d'AOS. Certains de ces travaux s'intéressent aussi à la mise en œuvre de l'adaptation dans le domaine des services Web, cependant, ils ne se préoccupent que d'une forme particulière d'adaptation (par exemple, seule l'adaptation par compensation à l'étape de surveillance est prise en compte). Or, nous souhaitons prendre en compte dans une même solution l'ensemble des problématiques sous-jacentes de l'AOS. Ceci demande de **couvrir une grande partie du cycle de vie de l'implémentation d'une telle application**. Cette transversalité permet ainsi d'offrir un cadre de travail homogène afin de **mettre en œuvre l'adaptation à différents niveaux du cycle de vie**.

1.3 Aperçu de la proposition

Notre proposition répond aux problématiques précédemment décrites à l'aide de deux modèles (un modèle de représentation de services Web et un modèle de composition de services Web) mis en œuvre à travers l'implémentation d'un registre de services Web et d'une plate-forme de génération d'applications adaptées au contexte d'utilisation à base de services Web.

Un modèle de représentation de services Web. Afin de faciliter les étapes de **recherche** et de **sélection**, la seule description fonctionnelle permise par le standard WSDL ne suffit pas. Nous proposons un modèle de représentation de services Web (WSR-Model, *Web Service Representation Model*) qui enrichit la **description** des services lors de leur **publication**. Cet enrichissement repose sur les catégories d'information suivantes :

- Le **domaine d'application** auquel appartiennent les services.
- Les **objectifs** du service.
- Les **aspects non fonctionnels** des services (tels que le fournisseur, les contraintes d'exécution).
- Les **aspects fonctionnels** des services (tels que les méthodes fournies, les paramètres d'entrées et de sorties).
- Le **contexte d'utilisation** auquel le service apporte de l'adaptation. Le fait que cette catégorie d'information appartienne au modèle de représentation de services Web permet d'intégrer l'adaptation au contexte d'utilisation au sein des étapes de **description** et de **publication**.

Un modèle de composition de services Web. Nous proposons un modèle de composition de services Web, nommé ProbCWS (*Problem-solving for Composition of Web Services*), basé sur la méthode de résolution de problèmes à base de tâches. Le problème à résoudre représente la composition à exécuter et les tâches les services Web composant la résolution du problème. Cette méthode a été choisie du fait qu'elle répond à un ensemble d'aspects relatifs à la composition de services Web :

- Ce type de méthode permet d'impliquer dans la définition de la **planification** de services Web la mise en œuvre de l'**adaptation au contexte d'utilisation**.
- Le moteur d'inférence qui exécute les méthodes de résolution de problème supporte la **surveillance** et ainsi permet de mettre en œuvre la **compensation** de services Web.

Un registre de services Web. Nous proposons un registre de services Web, nommé WSR, qui se base sur le modèle de représentation de services Web, WSR-Model, afin de proposer des fonctionnalités aux deux acteurs du domaine des services Web : les fournisseurs et les clients.

- WSR fournit un moyen aux fournisseurs de **publier** leurs services selon le modèle de représentation de services Web.
- WSR fournit trois méthodes de **recherche** aux clients :
    - Une méthode de recherche **par mots-clés**.
    - Une méthode de recherche **par catégorie de services**.
    - Une méthode de recherche **en vue de concevoir des applications adaptées au contexte d'utilisation**, basée sur les tâches à réaliser et le contexte d'utilisation pour lequel l'application doit être adaptée.

Une plate-forme de génération d'applications adaptées au contexte d'utilisation. Nous proposons une plate-forme de génération d'applications adaptées au contexte d'utilisation (nommée GenAWS – *Generating Adapted applications based on Web Services*) qui permet de mettre en œuvre la composition de services Web. L'utilisateur a la possibilité d'obtenir, soit la définition de la composition exprimée en ProbCWS, soit directement le résultat de l'invocation de l'application générée. Reposant sur le registre de services Web – WSR, et le modèle de composition – ProbCWS, GenAWS met en œuvre l'adaptation au contexte d'utilisation. Par conséquent dans GenAWS l'adaptation est réalisée au sein des étapes de planification et de surveillance du cycle de vie d'une application orientée services.

1.4 Organisation du document

Ce document est organisé en deux parties, décrivant respectivement l'état de l'art et notre proposition. La Figure 1.2 recense l'impact (analyse ou contribution) de chaque chapitre sur les étapes du cycle de vie des applications à base de services.

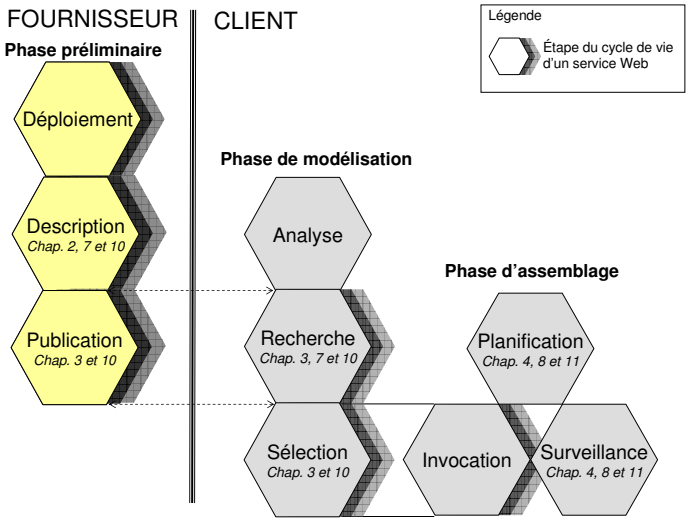


Figure 1.2 Correspondance des chapitres composant ce mémoire avec les étapes du cycle de vie d'une application à base d'architecture orientée services[6].

L'état de l'art porte sur les différentes étapes du cycle de vie d'une application à base d'architecture orientée services Web.

Le chapitre 2 présente les solutions existantes afin que les fournisseurs réalisent **l'étape de description** de leurs services Web. Dans un premier temps, le standard WSDL est étudié. Puis, les propositions intégrant de la sémantique aux descriptions de services Web sont exposées.

Le chapitre 3 décrit les travaux proposant aux fournisseurs de **publier** leurs services et aux clients de **rechercher** et **sélectionner** les services dont ils ont besoin. Nous décrivons tout d'abord les spécifications UDDI, puis les registres publics disponibles sur le Web. La dernière section de ce chapitre est consacrée à l'étude des travaux intégrant de la sémantique au sein de leur proposition de publication, recherche et sélection de services Web.

Les langages et plates-formes proposant de composer des services Web sont ensuite décrits dans le chapitre 4, en mettant en évidence les différents définitions et types de compositions dans le domaine des services Web. La composition de services Web correspond aux étapes de **planification** et **surveillance** du cycle de vie des applications à base d'AOS.

[6] Seules les étapes du cycle de vie pertinentes pour notre travail sont représentées dans cette figure.

Le chapitre 5 propose un état de l'art des travaux mettant en œuvre l'**adaptation** dans le domaine des services Web. Nous décrivons, dans un premier temps, les objectifs de l'adaptation et les définitions existantes du contexte. Ensuite, nous étudions les approches d'adaptation pour les services Web élémentaires, puis les approches d'adaptation pour la composition de services Web.

Le chapitre 6 analyse les travaux étudiés dans l'état de l'art afin de mettre en relief leurs apports et leurs manques. Par rapport à cette analyse nous fixons les objectifs et les axes d'étude de notre proposition. Cette synthèse de l'ensemble des travaux étudiés souligne :

- les différentes formes de **représentation de services Web** existantes ;
- les solutions en termes de **publication et de recherche de services Web** ;
- la mise en œuvre de la **composition de services Web** ;
- la prise en compte de l'**adaptation au contexte d'utilisation** dans le domaine des services Web.

La partie consacrée à notre proposition présente les deux modèles issus de nos travaux ainsi que les implémentations qui les mettent en œuvre.

Le chapitre 7 décrit le **modèle de représentation de services Web**. Ce modèle constitue les prémisses des étapes de **description** et de **publication** de services Web réalisées par les fournisseurs. L'originalité de notre approche repose, entre autres, sur le fait que nous proposons un moyen de **décrire le contexte d'utilisation** auquel le service représenté est adapté.

Le chapitre 8 est consacré à l'étude du **modèle de composition de services** que nous proposons. Ce modèle, nommé ProbCWS, se base sur les méthodes de résolution de problèmes à base de tâches et permet de mettre en œuvre les étapes de **planification** et de **surveillance** de la composition de services Web réalisées par les clients. À l'instar du modèle de représentation de services Web, le modèle de composition que nous proposons intègre la définition de la mise en œuvre de l'**adaptation au contexte d'utilisation**.

Le chapitre 9 met en évidence la mise en œuvre des deux modèles précédemment décrits. Nous avons choisi de faire reposer WSR-Model sur le système de représentation de connaissances par objets AROM et ProbCWS sur AROMTasks, le langage de résolution de problèmes par modèle de tâches associé à AROM.

Afin de permettre l'utilisation (et la réutilisation) des services Web, nous proposons **un registre de services Web**, **WSR**, qui repose sur les spécifications du modèle de représentation de services Web. Le chapitre 10 décrit ce registre et les fonctionnalités qu'il implémente :

- La **publication** de services Web par les fournisseurs.
- La **recherche** par les clients des services Web publiés.

L'**ensemble des étapes du cycle de vie** d'une application orientée services (de la publication d'un service Web élémentaire par le fournisseur à l'exécution de la composition de services Web) est contenu dans la plate-forme GenAWS étudiée dans le chapitre 11.

Enfin, le chapitre 12 dresse le bilan de cette thèse et présente les perspectives de ce travail.

ÉTAT DE L’ART

2 DESCRIPTION DE SERVICES WEB

L'un des nombreux objectifs de l'Architecture Orientée Services (AOS) est que les briques de base de l'implémentation (les services) puissent être réutilisées dans d'autres systèmes. La réutilisation de services Web repose sur le fait que ces derniers sont accessibles (sur le Web) et utilisables par le plus grand nombre de clients possibles (via un protocole standard). Ainsi, ce sont des services dits *universels* et non propriétaires. D'après [Nickul *et al.,* 2005], cette réutilisation est conditionnée par le fait que chaque service utilisé dans un système basé sur une AOS doit être, au préalable, décrit par son fournisseur. Les services Web étant largement utilisés pour implémenter des AOS [Erl, 2005], ces services doivent être décrits par leur fournisseur. Seconde étape du cycle de vie d'un service Web après son implémentation, la description d'un service Web est nécessaire pour deux raisons : la publication du service par son fournisseur dans un registre et sa sélection ultérieure par des clients via ce registre. Nous consacrons, par conséquent, le premier chapitre de l'état de l'art aux moyens qui existent aujourd'hui pour décrire les services Web.

Dans ce chapitre, nous nous préoccupons exclusivement de la description de services Web élémentaires puisqu'elle repose sur des langages spécifiques. Un service Web élémentaire est un service isolé, pouvant servir de services de base à une composition de services Web (*cf.* chapitre 4). Selon sa description, un service élémentaire peut être classique ou sémantique.

Service Web élémentaire classique. Un service Web est dit classique si sa description n'intègre que son (ou ses) profil(s) fonctionnel(s). Le profil fonctionnel intègre la description des informations permettant de faire appel au service (la description syntaxique des méthodes proposées, des paramètres de ces méthodes et des protocoles à utiliser).

Service Web élémentaire sémantique. Un service Web est dit sémantique si sa description intègre, en plus de la description fonctionnelle, une dimension sémantique. Cette dimension englobe toute description complémentaire (du fournisseur, des objectifs du service Web, etc.) de la description fonctionnelle qui permet d'ajouter de l'information à la description classique du service Web. Cette information peut ensuite être traitée par des mécanismes (tels que l'inférence) afin d'en extraire de la connaissance.

L'étude du standard de description de services Web classiques (WSDL) fait l'objet de la première section de ce chapitre, tandis que les travaux proposant une description de services Web sémantiques sont exposés dans la seconde section. Notons que les descriptions étudiées dans ce chapitre résultent d'une génération automatique. Afin de générer une description WSDL, il est possible d'utiliser des

plates-formes de développement telles que .NET[7] de Microsoft ou le projet AXIS[8] de la fondation Apache. Concernant la génération de OWL-S, des logiciels propres au Web sémantique tels que Protégé [Knublauch *et al.*, 2004] peuvent aider les concepteurs.

2.1 WSDL : le standard de description de services Web classiques

Nous étudions ici le standard de description des services Web proposé par le W3C : le langage WSDL. L'importance que nous attachons à l'étude de WSDL s'explique par le fait que WSDL est au cœur de la technologie des services Web. En effet, l'utilisation de WSDL pour décrire un service est un critère déterminant pour qualifier ce service de service Web.

Tout d'abord, nous étudions les concepts sous-jacents au standard WSDL en faisant abstraction de la structure et de l'encodage XML du langage. Ensuite, la structure et l'utilisation du standard sont présentées en décrivant les deux versions standardisées de WSDL (WSDL 1.1 [Christensen *et al.*, 2001] et WSDL 2.0 [Chinnici *et al.*, 2007]).

2.1.1 Concepts du langage WSDL

Les premiers travaux du W3C concernant un langage de description universelle de services Web ont vu le jour en 2001 lors de l'émergence de cette technologie. Il a fallu trouver un langage de description utilisable et compréhensible par le plus grand nombre afin que les services Web conçus soient interopérables. Le W3C a rapidement proposé le langage WSDL (*Web Service Description Language*) [Christensen *et al.*, 2001].

La description d'un service doit inclure la définition des composants nécessaires au protocole de communication (SOAP pour les services Web) et à l'interaction avec un client ou un autre service Web. Les problématiques de réutilisation et d'interaction ont guidé le W3C afin de définir les catégories d'information à prendre en compte dans la description d'un service Web. Les différents éléments décrits dans WSDL sont les suivants :

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

Dans WSDL, il existe une séparation entre deux niveaux indépendants, respectivement nommés abstrait et concret. Le *niveau abstrait* regroupe les informations pouvant être réutilisées (non spécifique à un service), tandis que le *niveau concret* est constitué de la description des protocoles d'accès au service Web (information particulière à un service). Le niveau abstrait est utilisé principalement lors du processus de sélection, tandis que le niveau concret est seulement utilisé lors de l'invocation des méthodes du service Web.

Le **niveau abstrait** décrit les informations propres aux méthodes proposées par le service, ainsi que les informations traitant des messages et des données échangés lors de l'invocation du service. Si deux services proposent les mêmes méthodes, le niveau abstrait de description WSDL peut être réutilisé. Ce niveau est composé des informations suivantes :

[7] http://www.microsoft.com/france/msdn/net/default.mspx

[8] http://ws.apache.org/axis

- **Les types de données**. Le document WSDL permet de décrire les types de données échangées. WSDL supporte les types élémentaires (tels que les entiers, les chaînes de caractères) et les types complexes. Si les données échangées possèdent une structure particulière (*i.e.* un type complexe), il est possible de les décrire par l'intermédiaire d'un schéma XML [Thompson *et al.*, 2004].
- **Les messages**. Un message correspond aux données qui sont véhiculées selon les méthodes invoquées. Chaque opération du service possède deux définitions de message : la première correspond à la requête et la seconde correspond à la réponse. La description d'un message contient le nom de l'élément en paramètre – d'entrée ou de sortie selon le type du message – et son type.
- **Les opérations**. Une opération représente une unité de travail, c'est-à-dire une méthode proposée par le service Web décrit. Chaque opération est identifiée par son nom.

Le **niveau concret** décrit la manière dont le client accède à un service en particulier, et, est de ce fait, non réutilisable (propre à un service unique). Les informations décrites dans le niveau concret sont les suivantes :

- **Le protocole de communication**. La description des protocoles de communication permet de définir le protocole à utiliser pour l'appel des méthodes du service. Si nécessaire, le document WSDL peut contenir autant de descriptions de protocole que d'opérations, étant donné que le protocole de communication peut être différent pour chaque opération du service décrit.
- **Les ports d'accès au service**. Dans un document WSDL, l'accès au service est défini par une collection de ports d'accès. Chaque port représente la localisation du service (*i.e.* son URL). Un même service peut être accessible sur plusieurs ports différents.

Les informations contenues dans WSDL constituent la description du profil fonctionnel du service. Avec WSDL, le client peut invoquer le service par le biais de sa description abstraite (méthodes disponibles, paramètres d'entrée et sortie) et concrète (description des protocoles de communication et des points d'accès du service).

2.1.2 Structure du langage

Cette section a pour objectif de décrire la structure d'un document WSDL et d'illustrer son utilisation. Nous choisissons d'étudier les deux versions existantes de WSDL (les versions 1.1 et 2.0) puisque la plupart des services Web existants sont décrits à l'aide de la première version bien que la seconde soit standardisée par le W3C. De plus, les travaux étudiés dans la suite du mémoire se basent sur l'une et l'autre des versions.

Quelle que soit la version de WSDL, ce standard se base sur le langage semi-structuré XML [Bray *et al.*, 2006]. Étant donné que XML facilite l'interopérabilité des systèmes distants et qu'il est facilement extensible, WSDL a été rapidement adopté par les concepteurs de services Web.

L'étude de chacune des versions de WSDL repose sur la présentation de sa structure générale et d'un exemple de description de service Web illustrant son utilisation. L'illustration que nous avons choisie s'appuie sur la description d'un service Web nommé *GlobalWeather* proposé par le fournisseur de service Web *webserviceX.net*[9]. Selon la ville et le pays, ce service Web renvoie un ensemble d'informations concernant la météorologie (vitesse du vent, visibilité, condition météorologique, température, etc.).

[9] http://www.webservicex.net

2.1.2.1 WSDL 1.1

La première version de WSDL assez aboutie pour être utilisée par les concepteurs de services Web, est la version 1.1 [Christensen *et al.*, 2001], issue des travaux du consortium W3C. Cette version a été rapidement utilisée par les concepteurs de services Web, et les services Web disponibles aujourd'hui reposent dans l'ensemble sur cette version.

Un document WSDL est un document XML composé d'un élément racine (`definitions`) et de cinq sous-éléments obligatoires (`types`, `message`, `portType`, `binding`, `service`). La Figure 2.1 illustre la structure générale d'un document WSDL 1.1 et établit le lien entre les éléments XML et les catégories d'information définies dans la section précédente (section 2.1.1).

```
<definitions >
    <types/>        → Description des types de données structurés
    <message/>      → Description des messages                      } Niveau abstrait
    <portType/>     → Description de l'ensemble des opérations
    <binding/>      → Description des protocoles de communication   } Niveau concret
    <service/>      → Description des ports d'accès
</definitions>
```

Figure 2.1 Structure générale d'un document WSDL 1.1.

La suite de l'étude de WSDL 1.1 est composée de l'illustration de chaque élément principal composant un document WSDL (les éléments `definitions`, `types`, `message`, `portType`, `binding` et `service`). Cette illustration repose sur la description WSDL 1.1 du service Web *GlobalWeather*.

L'élément `definitions`. L'élément `definitions` (*cf.* Figure 2.2) est la racine du document WSDL. Les espaces de noms permettent de connaître la version de SOAP (ligne 3), les définitions de schémas XML (ligne 4), et d'autres espaces de noms à utiliser lors de l'appel du service Web décrit (lignes 2 et 5 à 8). Le fournisseur du service Web décrit est identifié par l'espace de noms le localisant (l'attribut `targetNamespace`, ligne 9). L'espace de noms `wsdl` (ligne 10) est, par défaut, l'espace de nom de tout élément ne possédant pas d'espace de noms dans un document WSDL.

```
1   <wsdl:definitions
2       xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
3       xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
4       xmlns:s="http://www.w3.org/2001/XMLSchema"
5       xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
6       xmlns:tns="http://www.webserviceX.net"
7       xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
8       xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
9       targetNamespace="http://www.webserviceX.NET"
10      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
11      ...
12  </wsdl:definitions>
```

Figure 2.2 Illustration de l'utilisation de l'élément `definitions` dans la description du service Web *GlobalWeather* en WSDL 1.1.

L'élément `types`. L'élément `types` définit les structures de données contenues dans les messages échangés lors de l'appel du service Web décrit. Dans l'exemple de ce service Web (*cf.* Figure 2.3), nous pouvons voir que les messages échangent quatre types structurés (`GetWeather`, `GetWeatherResponse`, `GetCitiesByCountry`, `GetCitiesByCountryResponse` – respectivement lignes 3, 11, 12 et 13) et un type simple nommé `string` de type `string` (ligne 14). D'après l'extrait du document WSDL, nous

pouvons savoir que le type complexe `GetWeather` est composé de deux éléments (nommés respectivement `CityName` et `CountryName`, lignes 6 et 7), tous deux de type `string`. Ce sont les types de paramètres d'entrée d'une des méthodes du service Web.

```
1  <wsdl:types>
2    <s:schema elementFormDefault="qualified" targetNamespace="http://www.webserviceX.NET">
3      <s:element name="GetWeather">
4        <s:complexType>
5          <s:sequence>
6            <s:element minOccurs="0" maxOccurs="1" name="CityName" type="s:string"/>
7            <s:element minOccurs="0" maxOccurs="1" name="CountryName" type="s:string"/>
8          </s:sequence>
9        </s:complexType>
10     </s:element>
11     <s:element name="GetWeatherResponse">...</s:element>
12     <s:element name="GetCitiesByCountry">...</s:element>
13     <s:element name="GetCitiesByCountryResponse">...</s:element>
14     <s:element name="string" nillable="true" type="s:string"/>
15   </s:schema>
16 </wsdl:types>
```

Figure 2.3 Illustration de l'utilisation de l'élément `types` dans la description du service Web *GlobalWeather* en WSDL 1.1.

L'élément `message`. L'élément `message` introduit les types de messages supportés par le service et décrit les données transmises lors des appels des méthodes du service Web. Dans la Figure 2.4, nous pouvons voir que dans le service Web *GlobalWeather*, quatre messages sont échangés entre le client du service et le service lui-même (`GetWeatherSoapIn`, `GetWeatherSoapOut`, `GetCitiesByCountrySoapIn`, `GetCitiesByCountrySoapOut` – respectivement lignes 1, 4, 5 et 6). Le message `GetWeatherSoapIn` a pour paramètre l'élément `GetWeather` défini dans l'élément `types` (*cf.* ligne 2).

```
1  <wsdl:message name="GetWeatherSoapIn">
2    <wsdl:part name="parameters" element="tns:GetWeather"/>
3  </wsdl:message>
4  <wsdl:message name="GetWeatherSoapOut">...</wsdl:message>
5  <wsdl:message name="GetCitiesByCountrySoapIn">...</wsdl:message>
6  <wsdl:message name="GetCitiesByCountrySoapOut">...</wsdl:message>
```

Figure 2.4 Illustration de l'utilisation de l'élément `message` dans la description du service Web *GlobalWeather* en WSDL 1.1.

L'élément `portType`. Cet élément décrit l'ensemble des opérations proposées par le service Web. La description WSDL d'un service Web prévoit un sous-élément `operation` pour chaque opération supportée par le service. La Figure 2.5 illustre la description des opérations proposées par le service Web *GlobalWeather*, pour le `portType GlobalWeatherSoap`, nommées respectivement `GetWeather` (ligne 2) et `GetCitiesByCountry` (ligne 6). L'opération `GetWeather` reçoit le message `GetWeatherSoapIn` lors de son appel (ligne 3) et renvoie le message `GetWeatherSoapOut` lors de sa réponse (ligne 4).

```
1  <wsdl:portType name="GlobalWeatherSoap">
2    <wsdl:operation name="GetWeather">
3      <wsdl:input message="tns:GetWeatherSoapIn"/>
4      <wsdl:output message="tns:GetWeatherSoapOut"/>
5    </wsdl:operation>
6    <wsdl:operation name="GetCitiesByCountry">...</wsdl:operation>
7  </wsdl:portType>
```

Figure 2.5 Illustration de l'utilisation des éléments `portType` et `operation` dans la description du service Web *GlobalWeather* en WSDL 1.1.

L'élément `binding`. L'élément `binding` définit les protocoles de communication et les spécifications des formats de données pour les ensembles d'opérations (ou `portType`). La Figure 2.6 illustre le fait que, pour l'opération `GetWeather` du `portType` `GlobalWeatherSoap`, le protocole de communication à utiliser est SOAP (ligne 2).

```
1  <wsdl:binding name="GlobalWeatherSoap" type="tns:GlobalWeatherSoap">
2    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
3    <wsdl:operation name="GetWeather">
4      <soap:operation soapAction="http://www.webserviceX.neti/GetWeather" style="document"/>
5      <wsdl:input>...</wsdl:input>
6      <wsdl:output>...</wsdl:output>
7    </wsdl:operation>
8    <wsdl:operation name="GetCitiesByCountry">...</wsdl:operation>
9  </wsdl:binding>
```

Figure 2.6 Illustration de l'utilisation de l'élément `binding` dans la description du service Web *GlobalWeather* en WSDL 1.1.

L'élément service. Cet élément décrit la collection des ports d'accès du service. Le but de cet élément est de pouvoir localiser le service Web proprement dit, et ainsi pouvoir faire appel aux méthodes disponibles. Dans l'exemple du service Web *GlobalWeather* (*cf.* Figure 2.7), le service est disponible à l'URL suivant : `"http://www.webservicex.com/globalweather.asmx"`[10] (*cf.* ligne 3).

```
1  <wsdl:service name="GlobalWeather">
2    <wsdl:port name="GlobalWeatherSoap" binding="tns:GlobalWeatherSoap">
3      <soap:address location="http://www.webservicex.com/globalweather.asmx"/>
4    </wsdl:port>
5  </wsdl:service>
```

Figure 2.7 Illustration de l'utilisation de l'élément `service` dans la description du service Web *GlobalWeather* en WSDL 1.1.

2.1.2.2 WSDL 2.0

En 2002, le consortium W3C inscrit une nouvelle activité de recherche dans ses préoccupations : l'activité sur les services Web (*Web Services Activity*)[11]. De cette activité découlent quatre groupes de travail dont le groupe de travail sur la description de services Web (*Web Services Description Working Group*)[12]. Ce groupe a été créé principalement en vue de standardiser WSDL qui n'est en 2002, date de fondation de ce groupe de travail, qu'une note (*cf.* section 2.1.2.1). En 2007, WSDL dans sa version 2.0 est standardisé [Chinnici *et al.*, 2007]. À la suite de cette standardisation, les études du groupe de travail sur la description de services Web ont pris fin, l'objectif final du groupe étant atteint.

À l'instar de WSDL 1.1, WSDL 2.0 repose sur le langage semi-structuré XML. Sa structure générale est composée d'un élément racine (`description`) et de quatre sous-éléments obligatoires (`types`, `interface`, `binding` et `service`). La Figure 2.8 illustre la structure générale d'un document WSDL 2.0 et établit le lien entre les éléments XML et les catégories d'information définies dans la section 2.1.1.

[10] Dans cet exemple, la localisation du service n'a pas pour extension wsdl mais asmx, étant donné que le service a été conçu par l'intermédiaire de la plate-forme de développement d'application Web ASP.NET (http://www.asp.net/) proposée par Microsoft et integrée à la plate-forme .NET.

[11] http://www.w3.org/2002/ws/

[12] http://www.w3.org/2002/ws/desc/

```
<description>
    <types/>                  → Description des types de données structurés
    <interface>               → Description de l'ensemble des opérations
        <operation>                                                          Niveau abstrait
            <input/>
            <output/>         → Description des messages
        </operation>
    </interface>
    <binding/>                → Description des protocoles de communication
    <service/>                → Description des ports d'accès
</description>
```

Figure 2.8 Structure générale d'un document WSDL 2.0.

Dans la suite de cette section, nous illustrons l'utilisation de chaque élément XML composant un document WSDL 2.0 (`description`, `types`, `interface`, `binding` et `service`) avec l'exemple du service Web *GlobalWeather*. De plus, nous établissons, lorsque cela est possible, les liens syntaxiques entre les versions 1.1 et 2.0 de WSDL.

L'élément `description`. Cet élément, élément racine d'un document WSDL 2.0, à l'instar de l'élément `definitions` de la version 1.1, est utilisé afin de déclarer les espaces de nom utilisés tout au long du document (*cf.* Figure 2.9). L'espace de nom `xmlns="http://www.w3.org/ns/wsdl"` (ligne 2) est l'espace de nom utilisé par défaut.

```
1  <description
2      xmlns="http://www.w3.org/ns/wsdl"
3      xmlns:tns="http://www.webserviceX.NET"
4      targetNamespace="http://www.webserviceX.NET">
5
6  </description>
```

Figure 2.9 Illustration de l'utilisation de l'élément `description` dans la description du service Web *GlobalWeather* en WSDL 2.0.

```
1  <types>
2     <s:schema xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
3               xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
4               xmlns:s="http://www.w3.org/2001/XMLSchema"
5               xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
6               xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
7               xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
8               xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
9               elementFormDefault="qualified"
10              targetNamespace="http://www.webserviceX.NET">
11       <s:element name="GetWeather">
12          <s:complexType>
13             <s:sequence>
14                <s:element minOccurs="0" maxOccurs="1" name="CityName" type="s:string"/>
15                <s:element minOccurs="0" maxOccurs="1" name="CountryName" type="s:string"/>
16             </s:sequence>
17          </s:complexType>
18       </s:element>
19       <s:element name="GetWeatherResponse">...</s:element>
20       <s:element name="GetCitiesByCountry">...</s:element>
21       <s:element name="GetCitiesByCountryResponse">...</s:element>
22       <s:element name="string" nillable="true" type="s:string"/>
23    </s:schema>
24 </types>
```

Figure 2.10 Illustration de l'utilisation de l'élément `types` dans la description du service Web *GlobalWeather* en WSDL 2.0.

L'élément `types`. Cet élément décrit les types de messages que le service envoie et reçoit lors de l'appel d'une des méthodes. Cet élément est analogue à l'élément homonyme utilisé dans la version 1.1, à la différence que des espaces de noms nécessaires à la définition de la structure de données sont inclus à ce niveau du document (tel que l'espace de nom xmlns:s="`http://schemas.w3.org/2001/XMLSchema`" – ligne 4). La Figure 2.10 montre que la définition de l'élément structuré `GetWeather` (lignes 11 à 18) est identique à celle utilisée dans la version 1.1 (*cf.* Figure 2.3).

L'élément `interface`. Cet élément décrit l'ensemble des fonctionnalités, appelées opérations, fournies par le service Web. Une opération (sous-élément `operation`) représente une simple interaction entre le client et le service. Dans la version 2.0 de WSDL les messages sont définis au niveau de l'opération et sont de deux types : soit des messages que le service reçoit (sous-élément `input`), soit des messages que le service envoie au client (sous-élément `output`). De plus, l'élément `operation` possède un attribut `pattern` qui définit la séquence selon laquelle les messages sont transmis. Dans l'exemple du service Web *GlobalWeather* (*cf.* Figure 2.11), l'interface `GlobalWeatherSoap` contient deux opérations (`GetWeather` et `GetCitiesCountry`). L'opération `GetWeather` est composée de deux messages échangés entre le service et le client dont les paramètres sont `GetWeather` et `GetWeatherResponse` (leur structure de données est définie dans l'élément `type` *cf.* Figure 2.10 lignes 11 à 18 pour la structure de données de l'opération `GetWeather`).

```
1   <interface name="GlobalWeatherSoap">
2     <operation name="GetWeather"
3       pattern="http://www.w3.org/ns/wsdl/in-out">
4         <input element="tns:GetWeather"/>
5         <output element="tns:GetWeatherResponse"/>
6     </operation>
7     <operation name="GetCitiesByCountry"
8       pattern="http://www.w3.org/ns/wsdl/in-out">
9         <input element="tns:GetCitiesByCountry"/>
10        <output element="tns:GetCitiesByCountryResponse"/>
11    </operation>
12  </interface>
```

Figure 2.11 Illustration de l'utilisation des éléments `interface` et `operation` dans la description du service Web *GlobalWeather* en WSDL 2.0.

L'élément `binding`. Cet élément décrit comment accéder au service. Son rôle est le même que l'élément du même nom dans la version 1.1. L'espace de nom définissant le protocole à utiliser (dans cet exemple SOAP *cf.* Figure 2.12, ligne 1) est défini comme attribut de cet élément.

```
1   <binding xmlns:wsoap="http://www.w3.org/ns/wsdl/soap"
2    name="GlobalWeatherSoap"
3    interface="tns:GlobalWeatherSoap">
4      <operation ref="tns:GetWeather"  wsoap:soapAction="http://www.webserviceX.NET/GetWeather"/>
5      <operation ref="tns:GetCitiesByCountry"
6                  wsoap:soapAction="http://www.webserviceX.NET/GetCitiesByCountry"/>
7   </binding>
```

Figure 2.12 Illustration de l'utilisation de l'élément `binding` dans la description du service Web *GlobalWeather* en WSDL 2.0.

L'élément `service`. Cet élément définit la localisation du service Web décrit. Pour chaque interface décrite, un élément `service` lui est associé. Le sous-élément `endpoint` définit un port d'accès en référençant l'élément `binding` associé et en déclarant l'URL localisant le service (avec l'attribut `address`). Ceci permet qu'une interface d'un service possède plus d'une localisation (*i.e.* plus d'un élément `endpoint`) pour répondre aux problèmes d'indisponibilité.

L'exemple de l'élément `service` de la description du service Web *GlobalWeather* est illustré par la Figure 2.13.

```
1   <service name="GlobalWeather"
2   interface="tns:GlobalWeatherSoap">
3     <endpoint name="GlobalWeatherSoap" binding="tns:GlobalWeatherSoap"
4               address="http://www.webservicex.com/globalweather.asmx"/>
5   </service>
```

Figure 2.13 Illustration de l'utilisation de l'élément `service` dans la description du service Web *GlobalWeather* en WSDL 2.0.

2.1.2.3 Différences entre WSDL 1.1 et WSDL 2.0

Les concepts inhérents à WSDL (catégories d'information à prendre en compte et niveaux d'abstraction) sont présents dans les versions 1.1 et 2.0. Trois éléments du document WSDL (`types`, `binding` et `service`) sont identiques en termes d'informations décrites (respectivement type de données structurées, protocole de communication et ports d'accès) – *cf.* Tableau 2.1.

	Concepts	WSDL 1.1	WSDL 2.0
Niveau abstrait	Types de données	<types/>	<types/>
	Opérations	<portType/>	<interface> <operation/> </interface>
	Messages	<message/>	<input/> <output/>
Niveau concret	Protocole de communication	<binding/>	<binding/>
	Ports d'accès	<service/>	<service/>

Tableau 2.1 Comparaison des éléments XML utilisés dans chacune des versions de WSDL pour décrire les concepts de description de services.

Il existe cependant deux différences notoires entre les versions 1.1 et 2.0. La première différence, qui se situe au niveau structurel du langage, concerne la description des messages échangés entre le service et le client lors de l'appel des opérations disponibles. La seconde différence concerne le niveau d'extensibilité du langage.

La description des messages. Dans la version de WSDL 1.1, la description des messages repose sur l'élément `message`, alors que dans la version 2.0 elle se situe dans l'élément `operation` (lui-même sous-élément d'`interface`). La version 2.0 décrit de manière plus lisible les messages. En effet, la description des opérations et celle des messages se situent toutes deux dans l'élément `interface`, alors que dans la version 1.1 le message est décrit par un élément à part entière (l'élément `message`), et le lien entre l'opération et les messages se fait par un référencement dans l'élément `operation` (sous-élément de `portType`). Cette modification semble faciliter la lisibilité de la description du service Web.

L'extensibilité de WSDL. Dans les spécifications de WSDL 1.1, il n'existe pas de mécanisme d'extensibilité permettant d'ajouter des attributs aux éléments existants. Cette limitation a été levée dans la version 2.0. Cette possibilité d'extension est mise en pratique par la recommandation du W3C portant sur l'annotation sémantique de WSDL – SAWSDL (*cf.* section 2.2.2.2). SAWSDL utilise l'extensibilité de WSDL 2.0 afin d'ajouter de l'information (par

exemple, un paramètre de type `float` qui représente un prix est associé à une devise) aux propriétés fonctionnelles des services décrites par ce standard.

Des différences certaines existent entre les versions 1.1 et 2.0, non seulement sur le plan structurel mais aussi sur le plan de l'utilisation des services Web. La recommandation WSDL 2.0 est certes plus compréhensible et offre davantage de flexibilité d'utilisation du service Web. Néanmoins, aujourd'hui, les services Web existants sont décrits par les spécifications WSDL 1.1 et les plates-formes d'applications Web (telles que .NET) génèrent des descriptions WSDL 1.1. Le W3C et la fondation Apache offrent des outils permettant de convertir les descriptions de services Web relevant de la version 1.1 en WSDL 2.0 et de gérer (analyser et valider) les documents WSDL 2.0 (convertisseur de version de W3C[13] et le projet Woden d'Apache[14]). Cependant, à notre connaissance, aucun projet n'a pleinement adopté la version 2.0 à ce jour.

2.2 Description de services Web sémantiques

Quelles que soient les versions (1.1 ou 2.0) de WSDL, la description du service reste uniquement au niveau fonctionnel, c'est-à-dire qu'elle contient la manière dont on peut utiliser le service et non ce que fait le service. Par conséquent, la description WSDL reste insuffisante lors du processus de sélection. Des travaux proposent des moyens de décrire des services Web sémantiques afin de pallier cette difficulté et pour mettre en œuvre d'autres aspects tels que la découverte automatique de services.

Cette section présente tout d'abord la définition des services Web sémantiques, puis étudie les langages émergeants qui permettent de décrire ce type de services Web.

2.2.1 Définition des services Web sémantiques

L'objectif premier du Web sémantique est de définir et lier les ressources du Web afin de simplifier leur utilisation, leur découverte, leur intégration et leur réutilisation dans le plus grand nombre d'applications [Berners-Lee *et al.*, 2001]. Le Web sémantique doit fournir l'accès à ces ressources par l'intermédiaire de descriptions sémantiques exploitables et compréhensibles par des machines. Cette description repose sur des ontologies. Selon [Gruber, 1993], une ontologie est *une spécification explicite d'une conceptualisation*. Une *conceptualisation* est un modèle abstrait qui représente la manière dont les personnes conçoivent les choses réelles dans le monde et une *spécification explicite* signifie que les concepts et les relations d'un modèle abstrait reçoivent des noms et des définitions explicites [Gruber, 1993]. Le Web sémantique est devenu un domaine à part entière, preuve en est la création en 2001 du groupe de travail sur ce sujet par le W3C[15].

À ce jour, le standard de description WSDL ne supporte pas la description de services Web comme une ressource utilisable dans le contexte du Web sémantique. Or, l'automatisation des processus d'enregistrement, de recherche et d'acquisition peut faciliter, à terme, la tâche des concepteurs de systèmes à base de services Web qui doivent faire face à l'augmentation du nombre de services Web disponibles.

Afin d'automatiser les processus d'enregistrement (action d'identification du service Web), de recherche (action issue de la requête du client) et de sélection (action de choix) des services Web, des travaux académiques ont été initiés, principalement dans le domaine du Web sémantique [Payne *et al.*,

[13] http://www.w3.org/2006/02/WSDLConvert.html

[14] http://ws.apache.org/woden/

[15] http://www.w3.org/2001/sw/

2004]. Les services Web issus des travaux de ce domaine sont appelés des **services Web sémantiques** (par opposition à notre appellation de service Web classique). D'après [McIlraith *et al.*, 2003], les services Web sémantiques sont la combinaison de deux technologies (*cf.* Figure 2.14) : celle des services Web et celle du Web sémantique. Les services Web sémantiques sont des services Web dont la description est améliorée par des langages empruntés au Web sémantique, tel que RDF [Klyne *et al.*, 2004b] et OWL [McGuiness *et al.*, 2004]. Cet emprunt au Web sémantique permet à ces services Web d'être découverts et sélectionnés automatiquement par des machines ou d'autres services Web distants. Ceci permet aux services Web sélectionnés de répondre au mieux à la requête du client.

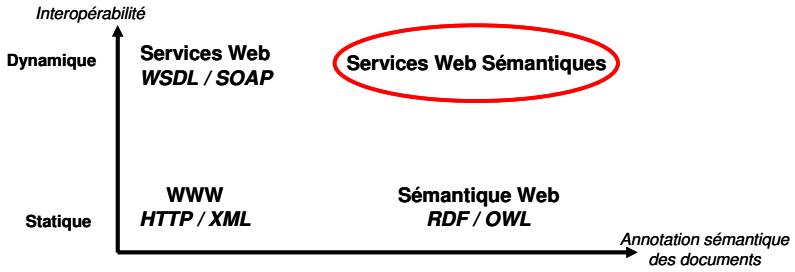


Figure 2.14 Les services Web sémantiques se situent entre deux problématiques liées au Web : l'interopérabilité et l'annotation sémantique des ressources, d'après [Fensel *et al.,* 2002a].

[Preist, 2007] introduit un ensemble de concepts-clés inhérents à cette technologie hybride que sont les services Web sémantiques. Nous n'évoquons ici que les concepts de *notion de service* et de *représentation de service* qui aident la compréhension des travaux portant sur la description de services Web sémantiques.

La notion de service. La notion de service par [Preist, 2007] intègre trois idées principales. Les deux premières respectent des notions sous-jacentes à la description d'un service Web classique : **un service est une interaction entre deux parties** (communément appelées dans le domaine des services Web *fournisseur* et *client*). **Un service doit être considéré à différents niveaux d'abstraction** : le service concret (suite spécifique d'actions) et le service abstrait (ensemble de services concrets sans les précisions techniques utiles lors de la sélection de services). La troisième idée sous-jacente à la notion de service est que cette dernière a un sens dans la mesure où **le service est inclus dans un domaine d'application** (par exemple, le domaine du tourisme). [Preist, 2007] définit ce domaine comme étant le *domaine de la valeur* du service.

La représentation du service. L'intérêt de combiner le Web sémantique et les services Web est de proposer une représentation intégrant les valeurs fournies par les services sous une forme compréhensible et interprétable automatiquement par les machines. Dans ce contexte, la description de services repose sur les techniques issues de la représentation de connaissances. Étant donné que ce domaine a développé des langages et techniques formels pour décrire les connaissances et raisonner sur ces dernières, la représentation de connaissances s'impose d'elle-même comme une solution pertinente. La description des services Web sémantiques repose sur deux choix. Tout d'abord, le choix du langage de représentation de connaissances à utiliser. Ensuite, le choix des concepts et relations à prendre en compte dans la description et leur(s) signification(s). Ceci implique la création d'ontologies ou la sélection d'ontologies existantes qui puissent fournir un vocabulaire ontologique structuré, c'est-à-dire un ensemble de concepts et de relations qui puissent être utilisés pour décrire des entités dans le domaine de la valeur [Preist, 2007].

2.2.2 Langages de description de services Web sémantiques

Des travaux, tels que [Fensel *et al.*, 2002b], [Roman *et al.*, 2005], proposent une manière de représenter de manière sémantique des services Web (autrement dit proposent de décrire des services Web sémantiques). Malgré l'abondance de ce type de travaux, aucun ne s'est imposé comme une solution de description de services Web sémantiques. Nous avons choisi d'aborder ici seulement les travaux issus du W3C qui tentent d'apporter une solution standard en termes de description de services Web sémantiques. Deux travaux sont présentés : OWL-S et SAWSDL.

2.2.2.1 OWL-S

OWL-S [Martin *et al.*, 2004] (*Ontology Web Language for Services*) est un langage issu des travaux de la DARPA[16] et de son programme *Agent Markup Language* (DAML) et prend la suite de DAML-S (*DARPA Agent Markup Language Service*)[17]. Il a été intégré au consortium W3C en 2004, au sein du groupe d'intérêt sur les services Web sémantiques[18], lors de la recommandation du langage OWL.

Le but initial du langage OWL-S est de mettre en œuvre des services Web sémantiques. Cette mise en œuvre inclut un grand nombre d'objectifs, rendus possibles par le biais de l'expressivité héritée de OWL et de l'utilisation de la logique de description [Baader *et al.*, 2003]. Ces objectifs sont :

- **la description de services Web sémantiques**. Cet objectif est étudié dans ce chapitre ;
- **la découverte automatique** de ces services (non prise en compte à ce jour) ;
- **l'invocation automatique** de ces services, par le biais de la détection et de l'interprétation automatique de la localisation et des paramètres d'entrée/sortie (étant donné que cet objectif a déjà été atteint par le standard de description de services Web classiques – WSDL, nous ne traitons pas ici cette partie du langage OWL-S) ;
- **la composition automatique** de services (description et invocation) et **la surveillance** de l'exécution de la composition. La composition de services Web via OWL-S est étudiée dans le chapitre 4 portant sur cette thématique.

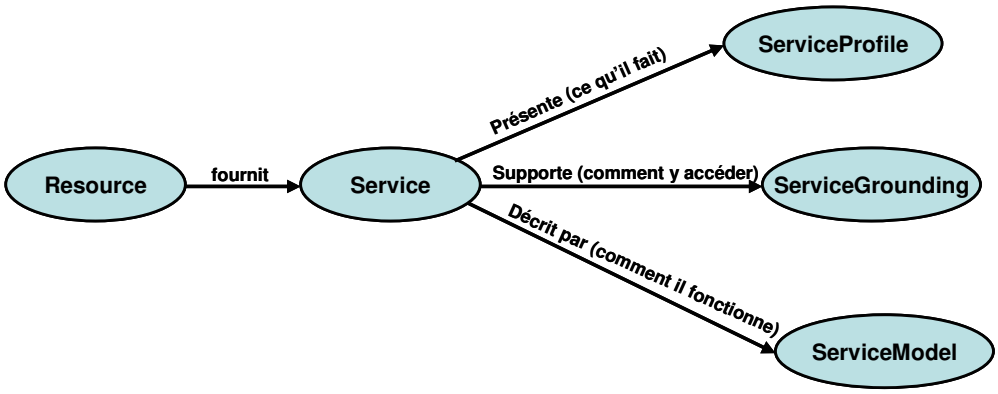


Figure 2.15 Diagramme fonctionnel de l'ontologie de service, d'après [Martin *et al.*, 2004].

La Figure 2.15 illustre le fait qu'une ressource fournit un service. Ce service présente un profil de service (*ServiceProfile*), est décrit par un modèle de service (*ServiceModel*) et supporte un accès de service (*ServiceGrounding*).

[16] DARPA est l'acronyme de Defence Advanced Research Projects Agency http://www.darpa.mil/

[17] http://www.daml.org/services/

[18] http://www.w3.org/2002/ws/swsig/

La description sémantique en OWL-S du service Web *GlobalWeather*, précédemment décrit par le standard WSDL (*cf.* section 2.1.2), est présentée par la Figure 2.16. La description OWL-S de ce service a été générée par le convertisseur WSDL2OWL-S, proposé par [Paolucci *et al.*, 2003] et disponible sur le site *SemWebCentral*[19].

```
1  <!DOCTYPE uridef[
2      <!ENTITY service "http://www.daml.org/services/owl-s/1.0/Service.owl">
3      <!ENTITY my_profile "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/profile.owl">
4      <!ENTITY my_process "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/process.owl">
5      <!ENTITY my_grounding "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/grounding.owl">
6  ]>
7  <rdf:RDF
8      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns"
9      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema"
10     xmlns:owl="http://www.w3.org/2002/07/owl"
11     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
12     xmlns:service="http://www.daml.org/services/owl-s/1.1/Service.owl"
13     xml:base="http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/service.owl">
14     <owl:Ontology about="">
15         <owl:imports rdf:resource="&service;" />
16         <owl:imports rdf:resource="&my_process;" />
17         <owl:imports rdf:resource="&my_profile;" />
18         <owl:imports rdf:resource="&my_grounding;" />
19     </owl:Ontology>
20     <service:Service rdf:ID="GlobalWeather">
21         <service:presents rdf:resource="&my_profile;#GlobalWeatherProfile"/>
22         <service:describedBy rdf:resource="&my_process;#GlobalWeatherProcess"/>
23         <service:supports rdf:resource="&my_grounding;#WsdlGrounding"/>
24     </service:Service>
25 </rdf:RDF>
```

Figure 2.16 Ontologie *Service* représentant l'ensemble des éléments composant la description sémantique du service Web *GlobalWeather* par le biais de OWL-S.

L'ontologie *Service* est l'élément principal de la description d'un service Web sémantique (*cf.* Figure 2.16). Elle référence les trois autres parties constituant la description. Tous les espaces de noms nécessaires sont décrits dans l'élément racine `rdf`. L'URL de la localisation de cette ontologie se situe dans cet élément racine (*cf.* ligne 13). L'élément `service` présente les éléments formant la description OWL-S du service *GlobalWeather* : le profil du service (`GlobalWeatherProfile`, lignes 3 et 21), le processus (`GlobalWeatherProcess`, lignes 4 et 22) et l'accès au service (`GlobalWeatherGrounding`, lignes 5 et 23).

L'élément *Profile* apporte une description du service et de son fournisseur (description abstraite du service). Le *ServiceProfile* du service est utilisé lors de la publication et la recherche d'un service. Il inclut trois types d'informations (décrites au format RDF) : le fournisseur, le comportement fonctionnel et les attributs fonctionnels. Le *ServiceProfile* du service *GlobalWeather* est illustré par la Figure 2.17.

Le fournisseur. Cette description est une représentation abstraite des acteurs intervenant dans la vie du service Web (le fournisseur et le client de service). Une liste d'informations permettant d'entrer en contact avec le fournisseur compose cet élément de description du service Web (telle que son adresse physique, l'URL du site Internet, son nom, son téléphone, son adresse électronique, son fax ou encore le contact de prestataires intervenant pour la maintenance du service). Dans le *ServiceProfile* du *GlobalWeather*, le fournisseur est défini par l'instance `WebServiceX` de la classe `Actor` (`<profile:Actor rdf:ID="WebServiceX"/>`, *cf.* ligne 17 de la Figure 2.17).

[19] SemWebCentral (http://projects.semwebcentral.org/) est un site Web de développement open source pour le Web sémantique créé en 2004.

```
1  <!DOCTYPE uridef[
2  <!ENTITY concept "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/GlobalWeatherConcept.owl">
3  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
4  ]>
5  <rdf:RDF
6      xmlns:process= "http://www.daml.org/services/owl-s/1.1/Process.owl"
7      xmlns:profile= "http://www.daml.org/services/owl-s/1.1/Profile.owl"
8      xml:base= "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/profile.owl"
9      ...
10     >
11     ...
12     <profile:Profile  rdf:ID="AddServiceName">
13         <profile:serviceName> GlobalWeather </profile:serviceName>
14         <profile:textDescription>  Get weather report for all major cities around the world.
15         </profile:textDescription>
16         <profile:contactInformation>
17             <profile:Actor rdf:ID="WebServiceX"/>
18         </profile:contactInformation>
19         <profile:hasInput>
20             <process:Input rdf:ID="GlobalWeatherSoap_GetWeather_parameters_IN">
21                 <process:parameterType rdf:datatype="&xsd;#anyURI">
22                     &concept;#GetWeatherTypeDeclaration
23                 </process:parameterType>
24             </process:Input>
25         </profile:hasInput>
26         <profile:hasOutput>...</profile:hasOutput>
27         <profile:hasInput>...</profile:hasInput>
28         <profile:hasOutput>...</profile:hasOutput>
29     </profile:Profile>
30 </rdf:RDF>
```

Figure 2.17 ***ServiceProfile*** **du service Web** ***GlobalWeather*** **défini par l'élément `Profile`.**

Le comportement fonctionnel. Cet élément de description permet de rendre publiques les opérations disponibles que propose le service, ainsi que leurs paramètres d'entrée et de sortie. De plus, le *ServiceProfile* inclut la description des pré-conditions nécessaires à l'appel des opérations et les effets attendus à la suite de l'exécution du service. Le service *GlobalWeather* (*cf.* Figure 2.17) propose deux opérations (dont *GetWeather*), possédant chacune un paramètre d'entrée (`<profile:hasInput>`, *cf.* lignes 19 à 24 et ligne 27) et un paramètre de sortie (`<profile:hasOutput>`, *cf.* lignes 26 et 28). La définition des types de données est opérée dans l'ontologie *Concept* (lignes 21 à 23 dans la Figure 2.17). Dans la Figure 2.18 est illustrée la déclaration du type (`GetWeatherTypeDeclaration`) du paramètre d'entrée de *GetWeather* est constituée de deux éléments (`CityName` et `CountryName`, *cf.* respectivement lignes 4 à 7 et 8 à 11) de type *string*.

```
1   <owl:Class rdf:ID="GetWeatherTypeDeclaration">
2       <rdfs:subClassOf rdf:resource="&owl;#Thing"/>
3   </owl:Class>
4   <owl:DatatypeProperty rdf:ID="CityName">
5       <rdfs:range rdf:resource="&xsd;#string"/>
6       <rdfs:domain rdf:resource="#GetWeatherTypeDeclaration"/>
7   </owl:DatatypeProperty>
8   <owl:DatatypeProperty rdf:ID="CountryName">
9       <rdfs:range rdf:resource="&xsd;#string"/>
10      <rdfs:domain rdf:resource="#GetWeatherTypeDeclaration"/>
11  </owl:DatatypeProperty>
```

Figure 2.18 Extrait de l'ontologie ***Concept*** **pour la description des types de données du service Web** ***GlobalWeather*****.**

Les attributs fonctionnels. Ce dernier type d'information contenu dans le *ServiceProfile* apporte des informations supplémentaires concernant le service. Tout d'abord, on peut trouver la **catégorie du service** (par exemple, en utilisant le système de classification nord-américain

UNSPSC[20] qui encode les produits et les services pour leur utilisation commerciale sur le Web. Ce système a l'avantage de contribuer au Web sémantique en formalisant ces catégories en ontologies via OWL et RDF). Ensuite, le futur client du service peut trouver le **niveau de qualité** déterminé par le fournisseur du service. Ce niveau de qualité peut être représenté, par exemple, par une appréciation (telle que « bon », « digne de confiance » ou encore « temps de réponse rapide »). Enfin, la troisième catégorie d'information stockée dans le *ServiceProfile* est constituée de **tout autre information jugée pertinente par le fournisseur** (telle que le coût du service, sa localisation géographique, sa disponibilité) afin que son service soit sélectionné par de futurs clients. La description du service Web *GlobalWeather* ne contient pas ce type d'information.

Le *ServiceModel*, dans le contexte d'un service Web élémentaire, présente le fonctionnement du service et définit comment interagir avec ce dernier. Dans le cadre de la description d'une composition de services, le ServiceModel permet aussi de décrire l'activité du service dans une composition (ceci est étudié dans le chapitre 4). À l'instar du *ServiceProfile*, le *ServiceModel* apporte une description abstraite du service Web sémantique. Dans cette entité de OWL-S, le service est vu comme un processus (*Process*). Un processus est une spécification de la manière dont le client peut interagir avec le service. Il est composé d'un ensemble d'informations sur ce dernier : son nom, les participants à son exécution (un client simple ou d'autres services Web), ce qu'il fait, les conditions d'utilisation et ses effets, son résultat, ses paramètres (entrée, sortie). Il existe trois types de processus : atomique (qui possède un accès), simple (qui fournit une vue sur d'un autre processus), et composé (qui est décomposable en processus simple). Étant donné que, dans ce chapitre, nous ne nous préoccupons que de la description d'un service Web élémentaire (et non composé), nous n'étudions ici que le processus *atomique* (les deux autres processus sont présentés dans le chapitre 4 concernant la composition de services Web).

Un processus atomique est un processus qui peut être directement invoqué par l'intermédiaire d'un accès (utilisation du *GroundingProfile*). Le service *GlobalWeather* possède deux processus atomiques (dont `GlobalWeatherSoap_GetWeather` – *cf.* Figure 2.19, lignes 12 à 15). La description du processus atomique contient les paramètres d'entrée et sortie et fait référence à l'ontologie *Concept* (ligne 2) pour la définition de la structure des données.

```
1  <!DOCTYPE uridef[
2  <!ENTITY concept "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/GlobalWeatherConcept.owl">
3  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
4  ]>
5  <rdf:RDF
6      xmlns:concept= "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/GlobalWeatherConcept.owl"
7      xml:base= "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/process.owl" >
8      ...
9      <process:Input rdf:ID="GlobalWeatherSoap_GetWeather_parameters_IN">...</process:Input>
10     <process:Output rdf:ID="GlobalWeatherSoap_GetWeather_parameters_OUT">...</process:Output>
11
12     <process:AtomicProcess rdf:ID="GlobalWeatherSoap_GetWeather">
13         <process:hasInput rdf:resource="#GlobalWeatherSoap_GetWeather_parameters_IN"/>
14         <process:hasOutput rdf:resource="#GlobalWeatherSoap_GetWeather_parameters_OUT"/>
15     </process:AtomicProcess>
16
17     <process:Input rdf:ID="GlobalWeatherSoap_GetCitiesByCountry_parameters_IN">...</process:Input>
18     <process:Output rdf:ID="GlobalWeatherSoap_GetCitiesByCountry_parameters_OUT">...</process:Output>
19     <process:AtomicProcess rdf:ID="GlobalWeatherSoap_GetCitiesByCountry">...</process:AtomicProcess>
20 </rdf:RDF>
```

Figure 2.19 *ServiceModel* du service Web *GlobalWeather*.

Le *ServiceGrounding* (élément `grounding`) d'une description OWL-S d'un service Web apporte une description concrète du service. La définition de l'accès du service (protocole d'accès au service,

[20] UNSPSC – *Universal Standard Protocol and Services Classification*, http://www.unspsc.org/

format de messages, appel aux opérations et type de transport) est contenue dans le *ServiceGrounding*. Pour décrire ces informations, le *ServiceGrounding* fait référence à la description standard du service en WSDL. Le *ServiceGrounding* permet de concrétiser la définition du processus dans le *ServiceModel*.

```
1  <!DOCTYPE uridef[
2      <!ENTITY pm_file "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/process.owl">
3      <!ENTITY local_service "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/service.owl">
4      <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
5  ]>
6  <rdf:RDF
7      xml:service="http://www.daml.org/services/owl-s/1.1/Service.owl"
8      xml:base= "http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/grounding.owl"...>
9      ...
10     <grounding:WsdlGrounding rdf:ID="WsdlGrounding">
11         <service:supportedBy rdf:resource="&local_service;#GlobalWeather" />
12         <grounding:hasAtomicProcessGrounding rdf:resource="#GlobalWeatherSoap_GetWeather_Grounding"/>
13         <grounding:hasAtomicProcessGrounding rdf:resource="#GlobalWeatherSoap_GetCitiesByCountry_Grounding"/>
14     </grounding:WsdlGrounding>
15     <grounding:WsdlAtomicProcessGrounding rdf:ID="GlobalWeatherSoap_GetWeather_Grounding">
16         <grounding:owlsProcess rdf:resource="&pm_file;#GlobalWeatherSoap_GetWeather"/>
17         <grounding:wsdlOperation>
18             <grounding:WsdlOperationRef>
19                 <grounding:portType rdf:datatype="&xsd;#anyURI">...</grounding:portType>
20                 <grounding:operation rdf:datatype="&xsd;#anyURI">GetWeather</grounding:operation>
21             </grounding:WsdlOperationRef>
22         </grounding:wsdlOperation>
23         <grounding:wsdlInputMessage rdf:datatype="&xsd;#anyURI">...</grounding:wsdlInputMessage>
24         <grounding:wsdlInput>...</grounding:wsdlInput>
25         <grounding:wsdlOutputMessage rdf:datatype="&xsd;#anyURI">...</grounding:wsdlOutputMessage>
26         <grounding:wsdlOutput>...</grounding:wsdlOutput>
27         <grounding:wsdlDocument  rdf:datatype="&xsd;#anyURI">
28             http://www.webservicex.com/globalweather.asmx?wsdl#
29         </grounding:wsdlDocument>
30         <grounding:wsdlReference rdf:datatype="&xsd;#anyURI">...</grounding:wsdlReference>
31     </grounding:WsdlAtomicProcessGrounding>
32     <grounding:WsdlAtomicProcessGrounding rdf:ID="GlobalWeatherSoap_GetCitiesByCountry_Grounding">
33         ...
34     </grounding:WsdlAtomicProcessGrounding>
35 </rdf:RDF>
```

Figure 2.20 *ServiceGrounding* du service Web *GlobalWeather*.

La Figure 2.20 illustre l'accès au service *GlobalWeather* par l'intermédiaire de son *ServiceGrounding*. Le service *GlobalWeather* possède deux accès (`GlobalWeatherSoap_GetWeather_Grounding` et `GlobalWeatherSoap_GetCitiesByCountry_Grounding`) aux deux opérations du service (*cf.* respectivement lignes 15 à 31 et ligne 32). Ces accès aux services sont définis dans le document en les liant aux processus atomiques définis dans le *ServiceModel*. Cette liaison est possible via la description concrète par des références au document WSDL :

- vers les opérations (`<grounding:wsdlOperation>`, lignes 17 à 22),
- vers les messages échangés (`<grounding:wsdlInputMessage>` – ligne 23 et `<grounding:wsdlOutputMessage>` – ligne 25),
- vers les paramètres d'entrée et sortie (respectivement `<grounding:wsdlInput>` – ligne 24 et `<grounding:wsdlOutput>` – ligne 26).

La localisation de la description WSDL du service *GlobalWeather* est contenue dans le *ServiceGrounding* (`<grounding:wsdlDocument>`, lignes 27 à 29).

OWL-S permet à travers trois ontologies (*ServiceProfile*, *ServiceModel* et *ServiceGrounding*) de décrire de manière sémantique un service Web. Le standard de description WSDL est toujours utilisé en vue de décrire les aspects fonctionnels du service. OWL-S, bien qu'il ne soit pas standardisé par le W3C, commence à être largement utilisé dans les travaux du Web sémantique.

2.2.2.2 SAWSDL – Annotations sémantiques pour WSDL

À l'initiative du groupe de travail d'annotations sémantiques pour WSDL (*Semantic Annotations for WSDL Working Group*[21]), SAWSDL (*Semantic Annotations for WSDL and XML Schema*) [Farrell *et al.,* 2007] est une recommandation du W3C depuis août 2007. Ce groupe de travail prend part aux activités du W3C portant sur les services Web. L'objectif de SAWSDL est d'ajouter de la sémantique à la description WSDL des services et des schémas XML [Thompson *et al.*, 2004]. Dans la suite de ce mémoire, nous n'abordons pas l'enrichissement sémantique des schémas XML, bien que le concept reste proche de l'enrichissement de fichiers WSDL.

SAWSDL est la suite de WSDL-S (*Web Service Description Language – Semantic*) [Akkiraju *et al.,* 2005]. SAWSDL définit un mécanisme d'annotation en vue de spécifier les éléments de WSDL à l'aide d'ontologie(s). Cette annotation repose sur la définition d'attributs étendant le standard de description. Les annotations sémantiques référencent des ontologies pré-existantes. Le mécanisme d'annotation de SAWSDL est indépendant de tout langage de représentation d'ontologies.

SAWSDL propose deux sortes d'annotations sémantiques : une première pour identifier le concept sémantique (représentée par l'attribut `modelReference`) et une seconde pour faire le lien entre le concept et le document WSDL (représentée par les attributs `liftingSchemaMapping` et `loweringSchemaMapping`).

L'attribut `modelReference` permet d'annoter tous les éléments de WSDL (nous évoquons ici la version 2.0 de WSDL). Cependant, la recommandation de SAWSDL considère que l'annotation des éléments suivants apporte une signification particulière :

L'élément `interface`. Cet élément décrit l'ensemble des méthodes disponibles d'un service Web. L'annotation de cet élément apporte une définition sémantique de l'ensemble des méthodes proposées.

L'élément `operation`. L'élément `operation` décrit une méthode particulière. Cette annotation permet d'expliciter les effets des méthodes proposées par le service.

L'élément `fault`. Cet élément représente un message d'erreur lors de l'appel d'une méthode de l'interface. L'annotation de ce type d'élément donne un sens au message d'erreur.

L'élément `type`. Cet élément représente les structures de données des paramètres dans des messages échangés. Si cet élément requiert des spécifications exprimées en *XML Schema* pour définir des types particuliers (simple ou complexe), l'annotation sémantique de SAWSDL peut être aussi utilisée dans cet élément afin d'annoter sémantiquement ces structures de données.

Un attribut `modelReference` a pour valeur zéro, une ou plusieurs URI localisant des ontologies qui apportent de la sémantique aux éléments annotés. Ce type d'annotation est utilisé lors des processus de recherche et de sélection. Si le concept sémantique est décrit en format XML, le code peut être directement intégré dans le fichier WSDL. Nous avons choisi d'annoter sémantiquement le service Web *GlobalWeather*. La structure de données *GetWeatherResponse* est définie dans la description standard WSDL comme étant de type `string`. Cette chaîne de caractères englobe un ensemble d'informations concernant les observations météorologiques (telles que la température, l'humidité, la vitesse du vent). Nous avons choisi l'ontologie *weather-ont*[22] afin d'annoter le service Web *GlobalWeather*. La Figure 2.21 donne un extrait de cette ontologie, représentant la classe `WeatherObservation` utilisée dans l'exemple d'annotation SAWSDL décrite ci-dessous. La classe `WeatherObservation` possède quatre propriétés : `hasTemperature` (ligne 7), `hasHumidity`

[21] http://www.w3.org/2002/ws/sawsdl/

[22] Cette ontologie, proposée par le site SemWebCentral, est disponible à l'URL suivante : http://refapp.semwebcentral.org/weather/ont/weather-ont.owl

(ligne 8), `hasWindSpeed` (ligne 13) et `hasPrecipitation` (ligne 16). Dans l'illustration de l'ontologie nous remarquons que la propriété `HasWindSpeed` est représentée par le type `float` (ligne 22) et a pour unité le *miles* par heure (ligne 21).

```
1  <rdf:RDF (...)
2     xmlns="http://localhost/weather/ont/weather-ont.owl#"
3     xml:base="http://localhost/weather/ont/weather-ont.owl">
4     <!-- Weather Observation Class -->
5    <owl:Class rdf:ID="WeatherObservation"> (...)
6      <rdfs:subClassOf>
7        <owl:Restriction> <owl:onProperty rdf:resource="#hasTemperature"/> (...) </owl:Restriction>
8      </rdfs:subClassOf>
9      <rdfs:subClassOf>
10       <owl:Restriction> <owl:onProperty rdf:resource="#hasHumidity"/> (...) </owl:Restriction>
11     </rdfs:subClassOf>
12     <rdfs:subClassOf>
13       <owl:Restriction> <owl:onProperty rdf:resource="#hasWindSpeed"/> (...) </owl:Restriction>
14     </rdfs:subClassOf>
15     <rdfs:subClassOf>
16       <owl:Restriction> <owl:onProperty rdf:resource="#hasPrecipitation"/> (...) </owl:Restriction>
17     </rdfs:subClassOf>
18   </owl:Class>
19   <!-- Properties -->
20   <owl:ObjectProperty rdf:ID="hasWindSpeed">
21     <rdfs:label>Wind speed, mph.</rdfs:label>
22     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
23   </owl:ObjectProperty>
24 </rdf:RDF>
```

Figure 2.21 Extrait de l'ontologie *weather-ont*, illustrant la classe `WeatherObservation`.

La Figure 2.22 illustre l'annotation sémantique de la structure de données *GetWeatherResponse* (élément `GetWeatherResponse`, ligne 8), avec SAWSDL (attribut `sawsdl:modelReference`, ligne 9), par l'intermédiaire de l'item `WeatherObservation` de l'ontologie *weather-ont* (*cf.* ligne 9).

```
1  <description ... >
2      <rdf:RDF
3          xmlns:base="http://refapp.semwebcentral.org/weather/ont/weather-ont.owl">
4      </rdf:RDF>
5      <types>
6        <s:schema ... >
7          <s:element name="GetWeather"> ... </s:element>
8          <s:element name="GetWeatherResponse"
9          sawsdl:modelReference="http://refapp.semwebcentral.org/weather/ont/weather-ont.owl#WeatherObservation">
10             <s:complexType>
11               <s:sequence>
12                 <s:element minOccurs="0" maxOccurs="1" name="GetWeatherResult" type="s:string"/>
13               </s:sequence>
14             </s:complexType>
15         </s:element>
16         <s:element name="GetCitiesByCountry"> ... </s:element>
17         <s:element name="GetCitiesByCountryResponse"> ... </s:element>
18         <s:element name="string" nillable="true" type="s:string"/>
19       </s:schema>
20       ...
21     </types>
22     ...
23 </description>
```

Figure 2.22 Annotation sémantique de l'élément `type` de la description WSDL 2.0 du service *GlobalWeather*.

Deux autres attributs, nommés `liftingSchemaMapping` et `loweringSchemaMapping`, étendent le fichier WSDL afin de spécifier l'association entre la donnée sémantique (le concept) et l'élément WSDL à annoter et réciproquement. Ces mises en correspondance sont faites à l'aide de schémas XML (dont la localisation est donnée par la valeur de l'attribut). L'attribut `liftingSchemaMapping` associe aux éléments XML un modèle sémantique. Inversement, l'attribut `loweringSchemaMapping` associe au modèle sémantique une structure XML. Ces mises en

correspondance sont utiles, lors de l'invocation du service, lorsque la structure de données ne correspond pas de manière intuitive à l'organisation du modèle sémantique. L'utilisation de ces attributs (`liftingSchemaMapping` et `loweringSchemaMapping`) n'est pas illustrée ici du fait que dans notre travail nous ne nous préoccupons pas de la cohérence structurelle des données. Nous considérons que l'étape de sélection, préalable à l'étape d'invocation, vérifie la cohérence de la structure des données échangées.

Si le fichier WSDL à annoter répond aux spécifications de la version 1.1, l'annotation sémantique SAWSDL ne peut pas être directement appliquée comme dans la version 2.0 car l'extension d'attribut n'est pas permise. De ce fait, la spécification SAWSDL prévoit un élément `attrExtensions` de type complexe à ajouter en tant qu'élément fils des éléments à annoter. L'élément `attrExtensions` accepte les attributs `modelReference`, `liftingSchemaMapping` et `loweringSchemaMapping` afin de mettre en œuvre l'annotation SAWSDL. Les éléments de fichiers WSDL 1.1 pour lesquels SAWSDL admet une annotation sont les éléments suivants : `portType`, `part`, `fault`, `operation` (*cf.* paragraphe 2.1.2.1).

Si le service décrit en WSDL 1.1 admet une structure de données exprimée en *XML Schema*, l'annotation se fait de la même manière que celle décrite plus haut (*cf.* Figure 2.22), étant donné que *XML Schema* accepte l'extension attribut.

SAWSDL permet d'ajouter facilement une description sémantique aux descriptions WSDL déjà existantes. L'avantage de cette proposition est qu'elle permet de modifier des descriptions WSDL existantes sans trop alourdir le fichier de description. Cependant, SAWSDL ne permet pas d'annoter tous les éléments XML du document WSDL (par exemple, la description du fournisseur de service ne peut pas être annotée) et ne permet pas d'apporter de la sémantique aux éléments n'étant pas décrits par WSDL (tels que la qualité de service).

2.3 Conclusion

Dans ce chapitre, nous avons mis en relief les différents langages proposés par le consortium W3C permettant aux fournisseurs de décrire leurs services Web. Cette étape de description est le premier processus indispensable (après l'implémentation du service) dans le cycle de vie d'une application basée sur une AOS.

En termes de description de service Web élémentaire classique, le langage WSDL a su rapidement s'imposer, et est aujourd'hui incontournable au sein de la communauté des services Web. Cependant, pour rechercher et sélectionner des services Web dans le cadre de l'implémentation de systèmes à base de services, la description syntaxique du service proposée par WSDL seule ne suffit pas. De ce fait, de nombreux travaux ont proposé des solutions afin d'étendre WSDL pour y ajouter de nouvelles catégories d'information (tels que [Tsai *et al.*, 2002], [Sivashanmugam *et al.*, 2003], [Lopez-Velasco, 2005]).

L'intégration du Web sémantique dans la communauté des services Web (par le biais des langages sous-jacents au Web sémantique, tels que RDF et OWL) a apporté, entre autres, un niveau sémantique dans la description des services Web (on parle alors de services Web sémantiques) par le biais des langages OWL-S et SAWSDL. Cependant, la communauté des services Web n'a pas aujourd'hui adopté l'un ou l'autre de ces langages comme standard de description de services Web sémantiques.

En ce qui concerne OWL-S, nous pouvons mettre en évidence le fait que les acteurs du domaine de la composition de services Web et du Web sémantique ne sont pas encore à ce jour d'accord sur le rôle de OWL-S. Les acteurs du premier domaine pensent qu'il s'agit d'un langage de composition alors que les acteurs du domaine du Web sémantique voient plutôt OWL-S comme une ontologie de description. En effet, [Ankolekar *et al.,* 2004], dans une étude comparant OWL-S avec les différents

langages de la technologie des services Web, mettent en parallèle OWL-S tant avec des langages de description (tels que WSDL) qu'avec des langages de composition (tels que WS-CDL et BPEL4WS). De notre point de vue, l'utilisation de OWL-S aujourd'hui en tant que langage de description de services Web sémantiques, permettra, à terme, qu'il soit aussi utilisé en tant que langage de composition.

SAWSDL n'étant une recommandation que depuis août 2007, nous n'avons pas trouvé de retour d'expériences de son utilisation. D'après l'utilisation que nous avons fait de ce langage, nous pensons que les annotations sémantiques sont un moyen simple, mais limité, d'utiliser WSDL pour représenter des services Web sémantiques.

La description rend possible la réutilisation des services mais elle est aussi nécessaire lors des processus de publication et de sélection. Le fournisseur doit transmettre une description du service pour que le service soit visible sur un registre. Le client sélectionne, via le registre, le service dont la description lui convient le mieux. La recherche, la publication, et la sélection de services sont discutées dans le chapitre suivant.

3 PUBLICATION, RECHERCHE ET SÉLECTION DE SERVICES WEB

Une fois le service Web décrit par son fournisseur, pour qu'il puisse être utilisé (et réutilisé) par le plus grand nombre de clients, il doit être publié dans un registre public. Lors de la conception d'une application, ses concepteurs (futurs clients de services) doivent rechercher et sélectionner à partir de registres existants les services Web de leur choix. Dans ce chapitre, nous étudions ces trois étapes (publication, recherche et sélection) du cycle de vie d'un service Web, interdépendantes, bien qu'elles ne soient pas, en principe, réalisées par le même acteur.

Ce chapitre étudie tout d'abord UDDI qui est un ensemble de spécifications pour la mise en œuvre de registres pour les services Web, couramment utilisé dans l'implémentation de registres internes aux organisations. Il existe sur le Web un ensemble de registres publics qui propose aux fournisseurs de services Web des moyens de publier leurs services et qui offre aux futurs clients des moyens de rechercher et de sélectionner des services. La seconde section de ce chapitre répertorie les registres publics. Nous avons déterminé les registres à étudier selon leur(s) mécanisme(s) de recherche et de sélection. Dans le chapitre précédent, nous avons évoqué les avantages liés aux services Web sémantiques. Le processus de description n'est pas la seule étape dans laquelle sont intégrés les concepts et technologies du Web sémantique. La troisième section est consacrée aux travaux traitant de la publication, de la recherche, et de la sélection de services Web sémantiques.

3.1 UDDI : vers des spécifications standard de publication et recherche de services Web

Cette section étudie les spécifications UDDI[23] (*Universal Description Discovery and Integration*) [Clement *et al.*, 2004] proposées par le consortium OASIS[24] (*Organization for the Advancement of Structured Information Standards*) afin de publier et rechercher des services Web. Nous consacrons cette première section à ces spécifications du fait qu'elles sont les plus utilisées pour implémenter les registres internes aux organisations, et qu'elles sont souvent étendues par des travaux académiques qui

[23] http://uddi.xml.org/

[24] http://www.oasis-open.org/

ambitionnent de les appliquer à d'autres domaines tels que le Web sémantique [Sivashanmugam *et al.*, 2003].

UDDI a été conçu en 2000 à l'initiative d'un ensemble d'industriels (Ariba[25], IBM[26], Microsoft[27]), en vue de devenir le registre standard de la technologie des services Web. Pour convenir à la technologie des services Web, les services référencés dans UDDI sont accessibles par l'intermédiaire du protocole de communication SOAP, et la publication des informations concernant les fournisseurs et les services doit être spécifiée en XML afin que la recherche et l'utilisation soient faites de manière dynamique et automatique. UDDI constitue un méta-service possédant des fonctions de publication et de recherche. Nous étudions tout d'abord comment les spécifications UDDI rendent possible la publication des services Web. Ensuite, les moyens mis en œuvre pour rechercher et sélectionner des services Web sont décrits.

3.1.1 Publication de services Web avec UDDI

Les différents composants de la publication faite par UDDI sont des documents XML manipulant de l'information à propos du fournisseur (*Business Entity*), le service lui-même (*Business Service*), les accès au service (*Binding Template*), le type de service (*tModel*) et des relations entre deux parties (*Publisher Assertion*) (*cf.* Figure 3.1).

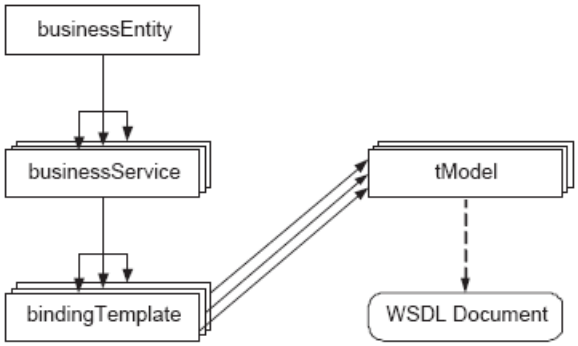


Figure 3.1 Entités composant un annuaire UDDI, d'après [Keidl *et al.*, 2004].

Le fournisseur (*Business Entity*). Les informations concernant l'organisation hébergeant le service, le fournisseur et celles nécessaires à l'identification de l'entreprise sont répertoriées dans ce document XML. Ce document contient de l'information descriptive sur l'entreprise ou le fournisseur et sur les services proposés (lien vers l'entité *Business Service* – *cf.* Figure 3.1).

Le service (*Business Service*). Ce composant représente les services proposés par l'organisation. La description des services contenue dans l'entité *Business Service* est de haut niveau (aucune information technique n'est décrite ici). Les informations à propos du nom du service et de son objectif sont représentées dans ce composant. Le fournisseur peut rassembler dans cette entité un ensemble de services répondant aux mêmes objectifs dans une même catégorie. Par exemple, une catégorie tourisme peut contenir un service météorologique et un service localisant les sites touristiques. Les catégories de service (contenant un ou plusieurs services) sont liées (selon le nombre de services) à un ou plusieurs points d'accès (*Binding Template* – *cf.* Figure 3.1).

[25] http://www.ariba.com/

[26] http://www.ibm.com

[27] http://research.microsoft.com

Les accès au service (*Binding Template*). Ce module décrit les points d'accès aux services Web (URL) et le moyen d'y accéder (les différents protocoles à utiliser) afin d'invoquer les services.

Le type de service (*tModel*). Le *tModel* permet d'associer un service à sa description WSDL. Le client potentiel peut ainsi avoir connaissance des conventions d'utilisation du service. La liaison entre les entités *Binding Template* et *tModel* est nécessaire pour l'invocation du service (*cf.* Figure 3.1).

3.1.2 Recherche de services Web avec UDDI

La recherche et la sélection dans UDDI reposent sur la publication préalablement décrite du service et de son fournisseur (*cf.* sous-section précédente). Le futur client peut connaître par l'intermédiaire de UDDI : les fournisseurs d'un service, les services proposés par un fournisseur donné, les moyens d'invoquer un service. Pour apporter aux clients la réponse à ces questions, UDDI organise l'ensemble des informations qu'il contient en trois parties, spécifiées en XML. Chacune d'elles peut être utilisée pour faire une recherche via UDDI. Ces parties sont les suivantes :

Les pages blanches (*White Paper*). Ce composant permet de connaître les informations à propos de l'organisation proposant le service. Cette description contient toutes les informations jugées pertinentes pour identifier l'organisation (telles que son nom, son adresse physique). Le futur client du service retrouve dans les pages blanches les informations que le fournisseur a renseignées dans l'élément *Business Entity* lors de la publication.

Les pages jaunes (*Yellow Paper*). Les pages jaunes de UDDI détaillent la description de l'organisation faite dans les pages blanches en répertoriant les services proposés. Dans cette section, sont décrits : la catégorie de l'entreprise, le secteur d'activité dans lequel exerce l'entreprise, les services offerts par cette organisation, le type de services et les conventions d'utilisation – prix, qualité de service, etc. Cette description repose sur les classifications standard de l'industrie nord américaine (NAICS[28] et UNSPSC[29]). La description des services contenue dans les pages jaunes est non technique et est renseignée par les fournisseurs eux-mêmes.

Les pages vertes (*Green Paper*). Les pages vertes comportent les informations techniques liées aux services Web et basées sur leur description WSDL.

À l'origine, il existait des registres UDDI dits publics (tels que ceux de Microsoft ou IBM) pour lesquels n'importe qui pouvait devenir, soit fournisseur, soit client de services Web. L'universalité de ces registres devait amener UDDI à devenir le standard de publication des services Web. En 2006, le nombre de services Web publiés a atteint le nombre de 50000. Malgré ce nombre, UDDI n'a jamais atteint son but : devenir le registre standard des services Web. Par conséquent, la maintenance des registres publics UDDI (tels que ceux de Microsoft et d'IBM) a été suspendue. Le réel succès de UDDI se situe au niveau des registres privés. En effet, de nombreuses organisations utilisent les spécifications de UDDI afin d'implémenter leur propre registre de services Web. Cependant, d'après [Dovey *et al.*, 2005], les spécifications UDDI souffrent de certaines limitations : il n'existe pas de plate-forme d'édition ; les API de UDDI sont insuffisantes pour développer efficacement des méthodes de publication et de recherche ; le modèle de recherche de UDDI est pauvre (recherche portant sur l'identifiant, le nom du service, ou sur les éléments du document WSDL).

[28] NAICS – *North America Industry Classification System*, http://www.naics.com

[29] UNSPCS – *Universal Standard Protocol and Services Classification*, http://unspcs.org

3.2 Registres de services Web classiques accessibles sur le Web

Les registres publics UDDI n'étant plus maintenus, différentes solutions de registres publics ont été créées et sont disponibles sur le Web. Ces registres ont vu le jour afin de proposer une interface entre les fournisseurs et les clients de services Web. L'analyse que nous menons sur les registres existants s'intéresse à trois aspects : la représentation des services Web proposée par les registres, les méthodes de recherche, et les critères de sélection proposés aux clients. Parmi le panel de registres accessibles via le Web (tels que *Web Service List*[30]), nous avons choisi d'en étudier deux (*XMethods*[31] et *RemoteMethods*[32]) issus des propositions d'industriels, et un travail académique (*QWS Dataset*[33]). Notre choix est orienté vers trois champs d'observation (la représentation de services, les méthodes de recherche, et les critères de sélection).

3.2.1 XMethods

Le site *XMethods*[31], proposé par l'organisation du même nom, est un registre de services Web hébergé sur Internet. Il contient environ 500 Web services[34]. Ce site permet de publier des services Web et de faire la recherche de services préalablement publiés.

La publication des services sur le site *XMethods* repose sur une publication UDDI. Le fournisseur doit tout d'abord être inscrit auprès du site *XMethods*. Ensuite, le fournisseur doit décrire un document *tModel* et un document *Business Service* selon les spécifications UDDI (*cf.* section 3.1.1) et une extension spécifique au site *XMethods*.

Le document tmodel. Ce document contient une liste de services Web que le fournisseur souhaite publier sur *XMethods*. L'ensemble des services est trié par le biais de leur description WSDL et de la localisation du document WSDL (identifiée par un URL). *XMethods* propose aux fournisseurs de décrire en langage naturel le service fourni (par le biais d'une description courte – une phrase, et d'une description détaillée – environ un paragraphe) et des notes d'utilisation à destination des futurs clients. Ces descriptions textuelles sont à ajouter dans le document *tModel* par le biais des éléments `<short description/>`, `<detailed description/>` et `<usage notes/>`, qui étendent les spécifications UDDI.

Le document Business Service. En plus des informations prises en compte dans la spécification UDDI, *XMethods* propose d'ajouter dans le document *Business Service* des informations concernant l'e-mail du fournisseur (élément `<contact email/>`), l'outil de développement (tel que .NET, AXIS) avec lequel le fournisseur a implémenté le service (élément `<implementation/>`). Les ports d'accès au service (décrits par le document *Binding Template* associé au *Business Service* et au *tModel*) doivent être renseignés par le fournisseur.

Pour rechercher un service publié sur *XMethods*, le client a deux listes à sa disposition. La première contient la liste des services Web récemment enregistrés et la seconde, dite complète, recense l'ensemble des services Web disponibles sur ce site (*cf.* Figure 3.2). Aucune méthode de recherche n'est disponible, les clients de services Web doivent chercher un service convenant à leurs besoins en explorant l'ensemble des services publiés. Les services Web apparaissant sur les listes sont ordonnés selon la date de mise en ligne, ce qui ne facilite pas le processus de recherche. Les critères

[30] http://www.webservicelist.com/

[31] http://xmethods.com/

[32] http://www.remotemethods.com/

[33] http://www.uoguelph.ca/~qmahmoud/qws/index.html

[34] Compte du nombre de services fait le 22/01/2009.

discriminants les services sont le fournisseur (*Publisher*), le nom du service (*Service Name*), la description du service (*Description*) et l'outil avec lequel le fournisseur a implémenté le service (*Implementation*) (*cf.* Figure 3.2).

Publisher	Style		Service Name	Description	Implementation
StrikeIron	DOC	Try It	Wall Street Horizon Real-Time Company Earnings	Information to analyze and evaluate investments and future earning potential	
SOATrader	RPC	Try It	Captcha Web Service	This Web service will create captcha requests.	
StrikeIron	DOC	Try It	StrikeIron Reverse Phone Business Intel	Get detailed information about a company based on a phone number	
StrikeIron	DOC	Try It	StrikeIron Reverse Phone Residential Intel	Get detailed information about a residence based on a phone number	
melmasry	DOC	Try It	Jobs in British Columbia and Alberta	Jobs in Western Canada	MS .NET
TQFTD2007	DOC	Try It	Mighty Maxims	Hand-picked selections from the The Quote For Today!	MS .NET
TQFTD2007	DOC	Try It	Mighty Meals	A refreshing Cookpedia.com recipe every day!	MS .NET
kyngas	DOC	Try It	Yellow Pages Lookup	search for companies by name	
StrikeIron	DOC	Try It	MapQuest Basic Mapping	Add mapping to your Web site or application	
cocoma		Try It	Cocoma Google Search Web Services	Check your search engine placement and page ranking.	
cocoma		Try It	Cocoma Video Web Services	Get video list from Google, MSN, Yahoo, AOL.	

Figure 3.2 Extrait de la liste des services Web disponibles sur *XMethods*.

Toutefois, afin d'aider la sélection des services Web contenus dans la liste, le client peut disposer d'informations supplémentaires sur le service telles que la localisation de la description WSDL, l'adresse de son site Web, et une description détaillée des tâches proposées par le service. Ces informations sont celles renseignées par le fournisseur lors du processus de publication. Elles sont visibles sur l'interface du site une fois que le client sélectionne le lien hypertexte du service.

Ce registre de services Web permet aux fournisseurs de rendre visibles leurs services Web via la publication sur un site Web. Cependant, du point de vue d'un client de service, *XMethods* ne propose aucune fonction de recherche et les catégories d'information décrivant le service reste basiques.

3.2.2 RemoteMethods

Le site *RemoteMethods*[35] est un registre de services Web accessible sur le Web créé en 2002. Il propose environ 400 services Web[36]. Ce site propose aux fournisseurs de publier leurs services et facilite les étapes de recherche et de sélection de services Web pour les clients.

RemoteMethods propose deux moyens de publier un service sur le site : le formulaire de publication ou la recommandation de service.

Le formulaire de publication. Le formulaire propose aux propriétaires de services de les rendre visibles sur *RemoteMethods*. La publication à l'aide du formulaire s'effectue en quatre étapes :

- L'inscription ou la validation du **profil de l'organisation** qui fournit le service.
- La publication de la **description détaillée du service** qui comporte des informations telles que le nom du service, le nom de la compagnie qui a développé le service, l'URL du site du fournisseur, la description WSDL, une description textuelle du service.
- La publication de la **description détaillée de l'évaluation du service** qui comporte des informations telles que le prix du service, la date de mise à jour du service, la durée de la période d'évaluation, si elle existe.

[35] http://www.remotemethods.com/

[36] Compte du nombre de services fait le 22/01/2009.

- Une fois le processus de description complété, la représentation du service est soumise à l'**évaluation des administrateurs de *RemoteMethods* pour approbation** de la publication.

La recommandation de services. Cette forme de publication permet à des clients de services non propriétaires de ces derniers de les recommander au site *RemoteMethods*. Cette recommandation repose sur un simple formulaire dans lequel les informations suivantes sont à renseigner :

- le nom du service à recommander,
- la page d'accueil du site du fournisseur,
- l'URL de la localisation de la description WSDL,
- la catégorie dans laquelle se situe le service (optionnelle),
- un commentaire textuel,
- l'adresse email de la personne qui recommande le service.

À la suite de la recommandation, les administrateurs de *RemoteMethods* décident de publier ou non le service recommandé et se chargent de décrire le service et son évaluation.

La recherche de services Web sur *RemoteMethods* repose sur un ensemble de huit catégories et sous-catégories de services Web pré-determinées (*cf.* Figure 3.3). Le client peut alors naviguer par le biais de liens hypertexte entre les catégories et sous-catégories proposées par *RemoteMethods*. Lorsqu'il n'existe plus de sous-catégories, les services Web relatifs à la (aux) catégorie(s) choisie(s) sont listés.

Figure 3.3 Catégories de services pour la recherche de services sur le site *RemoteMethods*.

Une fois que le client a trouvé la catégorie correspondant à ses besoins, il doit sélectionner le service qui lui convient le mieux. En vue d'aider les clients dans cette tâche, le site *RemoteMethods* met à la disposition du client un ensemble d'informations à propos du fournisseur, du service Web (renseigné, soit par le fournisseur, soit par le site lui-même), et un outil de tri de services.

Les informations de description du fournisseur. La description du fournisseur est composée d'informations sur la sécurité garantie par le fournisseur lors de l'invocation du service (avec les méthodes de sécurité utilisées) et d'informations sur les moyens mis en œuvre par le fournisseur pour garantir la maintenance de ses services Web (avec les horaires du support téléphonique, le temps de réponse à un mail, etc.).

Les informations de description du service renseignées par le fournisseur. Lors de la publication des services sur *RemoteMethods*, le fournisseur doit renseigner un ensemble d'informations qui permettent de faciliter le processus de sélection de son service. Ces informations sont le prix du service et une description textuelle de l'objectif du service. La date de mise à jour du service par le fournisseur est contenue dans la description du service.

Les informations de description du service renseignées par le site *RemoteMethods*. L'originalité de *RemoteMethods* est qu'il propose aux clients des services Web d'annoter les services Web qu'ils ont utilisés. Cette annotation est constituée d'une note globale (de une à cinq étoiles), d'un rapport d'erreurs (rapport textuel) et d'un compte-rendu. Le compte-rendu est composé des forces et faiblesses du service, d'un paragraphe de description libre, et de la durée d'utilisation du service sur laquelle se base l'évaluation du service.

Le tri de service. La liste de services Web peut être triée selon un ensemble de critères : par nom, par prix, par note, par date de mise à jour.

L'originalité de *RemoteMethods* repose sur l'aide à la navigation des clients dans l'ensemble des services Web disponibles et la possibilité, pour ces mêmes clients, de recommander des services Web. Bien que *RemoteMethods* offre une représentation riche des services publiés, la méthode de recherche proposée ne permet pas aux clients d'établir une requête portant sur l'ensemble des informations composant cette description.

3.2.3 Vers un registre centralisé de services Web classiques : QWS Dataset

QWS Dataset[37] (*Quality of Web Service Dataset*) [Al-Masri *et al.*, 2007a] [Al-Masri *et al.*, 2007b] [Al-Masri *et al.*, 2007c] est un projet académique mené à l'Université de Guelph au Canada par le département d'informatique. L'objectif de ce projet est d'offrir la possibilité aux chercheurs dans le domaine des services Web de trouver des services Web élémentaires en vue d'implémenter un système à base d'AOS. *QWS Dataset* contient 365 services Web[38].

Les services Web contenus dans *QWS Dataset* ne sont pas publiés par les fournisseurs mais récupérés par un moteur d'aspiration de services Web (*Web Service Crawler Engine*) [Al-Masri *et al.*, 2007a] à travers différentes sources telles que les registres UDDI, les moteurs de recherche ou les portails spécifiques aux services Web. Les services Web aspirés sont ensuite centralisés dans le *QWS Dataset*.

La recherche des services Web au sein de la liste disponible sur *QWS Dataset* repose sur des propriétés de qualité de service [Al-Masri *et al.*, 2007b]. Ceci permet de retrouver des services Web les mieux adaptés aux préférences du client en termes de qualité de service, de capacités des services Web et de caractéristiques des fournisseurs. Dans ces travaux, les définitions et mesures des propriétés de qualité de service sont celles données fréquemment dans le domaine de la QoS ([Ran, 2003] et [Kalepu *et al.*, 2003]) et incluent : le temps de réponse (en ms), le débit (en requête par minute), la disponibilité (en pourcentage), l'accessibilité (en pourcentage), l'analyse de l'interopérabilité (en pourcentage) et le coût (en cents par invocation).

Figure 3.4 Interface de recherche simple de *QWS Dataset*.

[37] http://www.uoguelph.ca/~qmahmoud/qws/index.html

[38] Compte des services fait le 22/01/2009.

QWS Dataset ne propose qu'une méthode de recherche disponible sur le Web qui est une simple recherche par mots-clés[39] (*cf.* Figure 3.4).

L'ensemble des services Web disponibles dans *QWS Dataset* est testé pendant une période de dix minutes, trois jours consécutifs, selon neuf attributs de qualité de service. Ces tests permettent d'attribuer un ensemble de caractéristiques portant sur la qualité de service [Al-Masri *et al.*, 2007c], utilisées par les clients lors du processus de sélection. Parmi cet ensemble de caractéristiques, six sont présentées à la suite d'une recherche pour discriminer les services Web retrouvés (*cf.* Figure 3.5) : le temps de réponse, le débit, la fiabilité, les meilleures pratiques, la documentation et le niveau de classification.

The QWS Dataset

Name	Response Time (ms)	Throughput (hits/sec)	Reliability (%)	Best Practices (%)	Documentation (%)	Class
FastWeather	125.44	13.5	86.4	80	91	★★★★
DOTSFastWeather	129.67	13.2	84.1	80	90	★★★★
WeatherForecast	261	1.8	58.1	80	94	★★★☆
WeatherFetcher	160	2.2	73.3	84	32	★★★☆
WeatherService	190.5	12.4	54	80	8	★★☆☆
GlobalWeather	1463.5	2.4	53.5	84	42	★★☆☆
ndfdXML	409.33	1.8	41.4	72	96	★☆☆☆

Figure 3.5 Résultat, dans *QWS Dataset*, de la recherche définie précédemment (*cf.* Figure 3.4) contenant les informations sur les services Web correspondant à la requête, utiles au processus de sélection.

Le temps de réponse (*Response Time*). Le temps de réponse est le temps mis entre l'envoi de la requête par le client et la réception du résultat envoyé par le service Web. L'unité de mesure du temps de réponse est la milliseconde (ms).

Le débit (*Throughput*). Le débit est mesuré par le nombre total d'invocations possibles d'un service Web dans un laps de temps donné. L'unité de mesure du débit est le nombre d'appels réussis par seconde.

La fiabilité (*Reliability*). La fiabilité est le rapport du nombre de messages d'erreur sur le nombre total de messages. Ce critère est mesuré en pourcentage.

Les meilleures pratiques (*Best Practices*). Le critère de meilleures pratiques évalue dans quelle mesure la description du service Web suit le WS-I BP (*WS-I Basic Profile*). Le WS-I BP [Ferris *et al.*, 2007] est une spécification proposée par le consortium d'industriel *WS Interoperability*[40] (WS-I). L'objectif de WS-I BP est de fournir un guide d'interopérabilité pour le noyau des spécifications des services Web (tel que SOAP, WSDL et UDDI), basé sur le format XML. WS-I BP est principalement utilisé dans le monde de l'industrie. Ce critère est mesuré en pourcentage.

La documentation (*Documentation*). La propriété QWS de documentation mesure pour le service Web sa capacité auto-descriptive selon l'examen du document WSDL. Cet examen consiste à vérifier s'il existe un contenu dans les balises décrivant le nom du service, les opérations du service, les messages échangés avec le client. L'unité de mesure de la documentation est en pourcentage.

[39] http://www.uoguelph.ca/~qmahmoud/qws/search.html

[40] http://www.ws-i.org/

Le niveau de classification (*class*). Le niveau de classification du service décrit la qualité offerte par le service (de la plus basse à la plus haute qualité représentée par zéro à quatre étoiles). Cette classification est basée sur l'ensemble des niveaux de qualité fourni par la méthode WsRF (*Web service Relevancy Function* [Al-Masri *et al.*, 2007b]) réalisée pour le projet.

L'avantage de ce travail est que tous les services Web qu'il propose sont des services qui fonctionnent. En effet, les services Web disponibles sont soumis à un test rigoureux. Ceci permet d'éviter de sélectionner un service qui n'est plus disponible. Cependant, les critères de description qui facilitent le processus de sélection, reposent principalement sur la qualité de service. Il manque une description de ce que propose le service. Le système d'aspirations de services Web, consistant à retrouver les fichiers de type wsdl sur le Web, est intéressant. En effet, il permet de rendre visibles les services Web disponibles sur le Web sans que les fournisseurs n'activent le processus de publication. Cependant, nous regrettons qu'il n'existe pas une possibilité aux fournisseurs de publier spécifiquement leurs services Web sur ce site.

La première section de ce chapitre a décrit UDDI, qui a été créé à l'origine dans le but de devenir le registre universel des services Web. UDDI ne répondant pas aux attentes des clients de services Web, des registres publics de services Web accessibles via le Web ont vu le jour. Dans la seconde section de ce chapitre nous avons étudié deux registres issus d'organisations (*Xmethods* et *RemoteMethods*) et un travail issu de travaux académiques (*QWS Dataset* [Al-Masri *et al.*, 2007a] [Al-Masri *et al.*, 2007b] [Al-Masri *et al.*, 2007c]). En vue de sélectionner les services publiés dans ces registres, ces derniers proposent d'enrichir la description des services. L'utilisation de langages issus du Web sémantique pour décrire les services Web peut encore faciliter la tâche du client lors des processus de recherche et de sélection de services Web (qui deviennent des services Web sémantiques). La section suivante présente des travaux portant sur la publication, la recherche et la sélection de services Web sémantiques.

3.3 Solutions de publication, de recherche et de sélection de services Web sémantiques

Cette section présente un ensemble de travaux qui prennent en compte les problématiques liées à la publication, la recherche et la sélection de services Web sémantiques. Nous abordons à travers quatre travaux les solutions existantes en termes de classification de services, d'annotation sémantique de services, de méthodes de raisonnement logique et de méthodes de résolution de problème en vue de rechercher et sélectionner les services Web sémantiques.

3.3.1 Découverte semi-automatique de services Web centrés utilisateur

L'objectif du premier travail étudié [Balke *et al.,* 2003b] est de découvrir des services Web dits centrés utilisateur. Cette découverte repose sur trois étapes : la définition d'une classification de services, la description sémantique des services, et un algorithme de découverte semi-automatique.

Dans [Balke *et al.,* 2003b], les auteurs définissent une classification de services Web selon trois critères : leurs interactions, leurs tâches et leur applicabilité.

Les interactions entre les acteurs. Les services Web peuvent répondre, soit à une interaction de type B2B (*business-to-business*, entre services), soit à une interaction de type B2C (*business-to-consumer*, entre un service et un utilisateur).

Les tâches des services et leur applicabilité. Dans le cas d'une interaction B2B, les tâches doivent être bien définies afin que les appels entre les services se fassent de manière automatique. Ce type d'interaction est applicable seulement si les pré et post-conditions correspondent. Dans le cas d'une interaction B2C, les tâches peuvent être : (1) des tâches simples applicables sans contraintes ; (2) des tâches complexes applicables seulement à l'aide d'informations complémentaires. Ces informations complémentaires (de type profil de l'utilisateur, préférences, etc.) permettent de découvrir les services Web adaptés aux utilisateurs.

La découverte de services Web dans [Balke *et al.,* 2003b] repose sur une description sémantique des services. Cette description est réalisée à l'aide du langage DAML-S [Ankolekar *et al.*, 2002] et du logiciel *OntoEdit* [Sure *et al*., 2002]. DAML-S est le langage précédant OWL-S (*cf.* chapitre 2). Les utilisateurs de DAML-S tentent d'employer le plus souvent possible des ontologies existantes pour décrire les services Web. Ces ontologies communes, appelées par les auteurs des *ontologies par défaut*, doivent, à terme, permettre à des agents logiciels ou des utilisateurs humains de découvrir, composer et invoquer des services Web.

Une fois les services Web décrits à l'aide de DAML-S, la découverte est mise en œuvre par un parcours sur les ontologies selon la requête de l'utilisateur (exprimée en SQL). Le parcours est terminé lorsqu'une instance de service Web correspondant à la requête est découverte. Afin d'améliorer la découverte de services, les auteurs intègrent des profils d'utilisateurs (description de l'utilisateur composée d'informations exactes sur le long terme, telles que son nom, sa date de naissance) et des patrons d'utilisation (représentant les préférences de l'utilisateur pour un cas d'utilisation particulier) [Balke *et al.,* 2003a]. Ces patrons représentent des descriptions anticipées d'utilisation. Afin de faire correspondre le processus de découverte aux patrons d'utilisation, d'autres ontologies sont conçues en intégrant ces descriptions. Il y a alors autant d'ontologies que d'anticipations d'utilisation.

De manière plus détaillée, l'algorithme de découverte, semi-automatique du fait que l'utilisateur intervient dans ce processus, est basé sur huit étapes :

Étape 1. Le système établit la correspondance entre la requête de l'utilisateur et les catégories de services.

Étape 2. Le système récupère les patrons d'utilisation existants correspondant à la requête de l'utilisateur.

Étape 3. Le système capture les intentions de l'utilisateur pour l'associer à un patron d'utilisation ou en créer un s'il n'en existe pas.

Étape 4. Le système choisit l'ontologie appropriée sur laquelle a lieu le processus de découverte.

Étape 5. Le système recherche un ensemble basique de services. Ce premier ensemble est, par la suite, filtré à l'aide d'informations supplémentaires.

Étape 6. Le système étend la requête en intégrant le profil de l'utilisateur [Balke *et al.,* 2003a] et retrouve de manière semi-automatique (avec l'interaction de l'utilisateur) les services Web à partir de ceux découverts dans l'étape 5.

Étape 7. Si l'utilisateur est satisfait par un service alors le système sort de l'algorithme.

Étape 8. Si l'utilisateur n'est pas satisfait alors le système élargit les caractéristiques requises du service dans l'ontologie choisie et retourne à l'étape 5.

L'originalité de ce travail est la mise en œuvre d'une découverte semi-automatique par laquelle le client est intégré au processus de recherche. Cependant, ceci ne permet pas d'utiliser cette proposition en l'état lorsque le client est un autre service Web. Le fait qu'il y ait autant d'ontologies que d'anticipations d'utilisation risque de compliquer la recherche. Enfin, le processus de découverte

proposé ici peut être assimilé à la recherche de services Web, à la différence que le corpus n'est pas limité à un registre (composé de services Web publiés par des fournisseurs), mais est ouvert au Web dans son intégralité.

3.3.2 ASSAM : outil d'annotation sémantique de services Web

Lors de la description des services Web sémantiques, des erreurs peuvent survenir étant donné qu'elle se fait manuellement. L'objectif de [Heß *et al.*¸2004] est de proposer un outil d'annotation sémantique automatique de services Web. Cet outil nommé ASSAM (*Automated Semantic Service Annotation with Machine learning*) est fait tant pour les clients que pour les fournisseurs. ASSAM apporte deux contributions : une application pour annoter les descriptions WSDL et un algorithme pour agréger les données.

Le processus d'annotation repose sur des méthodes d'apprentissage automatique (*Machine Learning*). Afin de mettre en œuvre cette annotation, les auteurs établissent une analogie entre la classification désirée des opérations et des données et les problèmes de classification de texte (*text classification problem*). Selon les annotations sémantiques connues d'autres services Web, l'algorithme d'apprentissage automatique peut généraliser et prédire des étiquettes sémantiques de nouveaux services Web. Les auteurs distinguent trois termes : la catégorie, le domaine et les types de données.

La catégorie. Ce terme dénote la sémantique du service Web.

Le domaine. Ce terme dénote la sémantique de l'opération.

Les types de données. Ce terme dénote la nature et le domaine de valeurs d'une variable.

Afin de mettre en œuvre la classification, les auteurs supposent qu'il existe une dépendance entre la catégorie, le domaine, et les types de données. Leur système de classification repose sur l'algorithme de classification itérative introduit par [Neville *et al*., 2003]. L'annotation concerne tout d'abord les services Web selon les catégories équivalentes, puis les opérations (éléments `portType` et `message` de WSDL) selon les domaines et enfin les paramètres selon les types de données. À partir de ces annotations sémantiques, ASSAM génère automatiquement le OWL-S associé à la description WSDL du service Web.

Afin de regrouper les opérations des services Web dont les paramètres d'entrée et de sortie sont équivalents, ASSAM intègre un outil d'agrégation : OATS (*Operation Aggregation Tool for Web Services*). L'OATS calcule la similarité entre un couple de paramètres selon leur distance sémantique. Une fois les distances calculées entre les paires d'éléments, l'outil d'agrégation regroupe les éléments semblables en se basant sur l'algorithme *Hierarchical Agglomerative Algorithme* (HAC) [Guedalia *et al.*¸1999]. L'HAC est un algorithme de catégorisation ascendante dans lequel les *clusters* peuvent posséder des sur-*clusters* et ainsi de suite.

Ce travail d'annotation sémantique de description de services Web et d'agrégation de services a l'avantage de se baser sur des algorithmes existants (tels que l'algorithme de classification itérative [Neville *et al.,* 2003] et l'algorithme HAC [Guedalia *et al.¸1999]*). Cependant, le fait d'annoter les ports type et les messages issus de la description WSDL est, de notre point de vue, critiquable. En effet, les ports type regroupent l'ensemble des opérations disponibles dans un service Web. Par conséquent, ce travail regroupe les services Web qui proposent rigoureusement le même ensemble d'opérations. De notre point de vue, l'annotation de l'élément `operation` est préférable afin que les groupes de services soient plus précis et facilitent la recherche et la sélection de services Web.

3.3.3 Approche basée sur une logique de description

Dans [Baader *et al.,* 2005], les auteurs proposent une représentation de service Web particulière formalisée à l'aide d'une logique de description. Les auteurs définissent un formalisme avec lequel on peut raisonner sur les descriptions de services Web, et ainsi rechercher les services par le biais d'inférences. Ce formalisme repose sur deux notions : les conditions et les interprétations.

Les conditions. Un service est un ensemble de pré et post-conditions représentées selon la notation suivante : `S=(pre, occ, post)`, où :

- `S` représente le service ;
- `pre` représente un ensemble fini de pré-conditions ;
- `occ` représente un ensemble fini de fermetures de la forme `A(a)` où `A` est un concept primitif et `a` son rôle. Le rôle des fermetures est de décrire les primitives pour lesquelles les conditions de minimisation ne sont pas appliquées.
- `post` représente un ensemble fini de post-conditions.

Les interprétations. Pour apporter de la sémantique aux services Web, les auteurs expriment comment l'application d'un service change le monde (*i.e.* comment le service change une interprétation I en une interprétation I'). Cet apport repose sur l'approche des modèles possibles (*Possible Model Approach* – PMA) proposée par [Winslett, 1988].

Afin de choisir le service Web qui convient le mieux, les auteurs introduisent deux tâches de raisonnement spécifiques aux services : leur exécutabilité et leur projection.

L'exécutabilité du service. L'expression de l'exécutabilité vérifie si le service peut être exécuté, c'est-à-dire si toutes les pré-conditions sont réunies pour invoquer le service.

La projection du service. L'expression de la projection d'un service vérifie si une condition est vraie après l'exécution du service.

En raisonnant avec ces tâches, le client sait s'il peut, avec les données dont il dispose, invoquer le service sélectionné (à l'aide de la définition de l'exécutabilité), et si le service sélectionné va produire le résultat désiré (à l'aide de la projection).

Concernant le formalisme, la description du service et les règles d'inférence sont réalisées dans [Baader *et al.,* 2005] à l'aide de $\mathcal{ALCQIO}$. $\mathcal{ALCQIO}$ est une extension du langage $\mathcal{AL}$ (*Attributive Language*), langage de définition syntaxique de base de la logique de description, introduit par [Schmidt-Schauβ *et al.*, 1989]. $\mathcal{ALCQIO}$ ajoute au langage $\mathcal{AL}$ l'expression des quantificateurs existentiels, de l'union des concepts, de la cardinalité, des propriétés inverses et des classes définies par extension.

L'utilisation d'une logique de description pour décrire les services Web est intéressante afin d'utiliser des langages pré-existants d'inférence pour mettre en œuvre les processus de recherche et de sélection. Cependant, ceci impose aux fournisseurs de services Web un travail lourd en termes de description. De plus, les deux tâches de raisonnement (exécutabilité et projection), sur lesquelles sont basées la recherche et la sélection, ne sont suffisantes d'un point de vue sémantique car il manque une catégorisation des objectifs des services.

3.3.4 Méthode de résolution de problèmes pour la sélection de services Web

Le projet IRS[41] (*Internet Reasoning Service*) du KMI[42] (*Knowledge Media Institute*) propose un cadre de travail pour la création d'applications à base de services Web sémantiques. Ce projet compte à ce jour trois versions. La première version IRS-I [Crubezy *et al.*, 2003] supporte l'implémentation de systèmes structurés à base de connaissances. La seconde version du projet IRS, IRS-II [Motta *et al.*, 2003], intègre à la première version les spécificités des technologies des services Web. Dans la troisième version, IRS-III [Cabral *et al.*, 2006], des ontologies WSMO – *Web Service Modeling Ontology* [Roman *et al.*, 2005] ont été introduites afin de permettre la description, la publication et l'exécution de services Web sémantiques.

WSMO est issu des travaux du consortium du même nom (ESSI-WSMO[43] groupe de travail de l'ESSI[44] – *European Semantic Systems Initiative*). La description d'un service Web sémantique réalisée à l'aide de WSMO possède quatre éléments : les ontologies du domaine, les objectifs, les services Web et les médiateurs.

Les ontologies du domaine. Ces ontologies fournissent les bases pour la description sémantique. L'objectif de cet élément est de permettre l'interopérabilité sémantique entre les différents éléments. De ce fait, il est utilisé par l'ensemble des parties constituant l'ontologie WSMO.

Les objectifs. Les descriptions des objectifs définissent les tâches du service qui peuvent répondre aux besoins du client.

Les services Web. Au sein de la description des services Web, les fournisseurs décrivent le comportement fonctionnel du service Web déployé. La description du comportement fonctionnel inclut les paramètres d'entrée et de sortie et les expressions logiques pour exprimer les contraintes d'utilisation.

Les médiateurs. Les médiateurs permettent de décrire les liens entre les éléments précédents (ontologies, objectifs et services Web).

Dans IRS-III, la publication des services Web repose sur l'annotation sémantique du service Web déployé. Par l'intermédiaire d'une plate-forme de publication propre à IRS, le fournisseur peut décrire l'ensemble des ontologies nécessaires à la description de son service en utilisant WSMO. Les ontologies sont ensuite associées au service Web déployé par le fournisseur. Lors du processus de publication, ce dernier doit aussi associer au service une méthode de résolution de problème.

La recherche et la sélection dans IRS-III a la particularité de se baser sur les méthodes de résolution de problème. Cette résolution est faite à l'aide du langage UPML (*Unified Problem Solving Method Development Language*) [Omelayenko *et al.*, 2003]. UPML distingue quatre notions : les modèles du domaine, les modèles de tâche, les méthodes de résolution de problème et les ponts. Chacune de ces notions est associée à une ontologie descriptive du service, présentée plus haut.

Modèles du domaine. Ces modèles décrivent le domaine d'application (par exemple, celui du tourisme). Ces modèles sont associés à l'élément WSMO *ontologie du domaine*.

Modèles de tâche. Ces modèles fournissent une description générique de la tâche à résoudre. Ces modèles sont associés à l'élément WSMO *objectif.*

[41] http://kmi.open.ac.uk/projects/irs/

[42] http://kmi.open.ac.uk/

[43] http://www.wsmo.org/

[44] http://www.essi-cluster.org/

Méthodes de résolution de problème. Cette notion fournit une description abstraite des processus de raisonnement qui permettent de résoudre des tâches dans des domaines spécifiques. Ces méthodes sont associées à l'élément WSMO *service Web.*

Ponts. Les ponts spécifient les relations entre les différents modèles d'une application. Cette notion est associée à l'élément WSMO *médiateur.*

Lorsque la plate-forme IRS-III reçoit une requête du client (équivalent à l'ontologie *objectif*) par l'intermédiaire d'une interface de communication, trois étapes se succèdent. La plate-forme : (1) recherche un ensemble de *services Web* pouvant répondre à l'*objectif* ; (2) sélectionne le plus approprié ; (3) l'invoque. Ces étapes reposent sur l'utilisation de l'ensemble des descriptions des services Web sémantiques qui sont composées des *objectifs*, des *services Web*, et des *médiateurs*. Ces ontologies sont supportées par les *ontologies du domaine* correspondant à la requête, selon un algorithme de correspondance.

Le cadre de travail IRS-III est intéressant puisqu'il prend en compte de manière transversale quatre processus sous-jacents à l'architecture des services Web (la description, la recherche, la sélection et l'invocation) via l'utilisation d'ontologies WSMO. De notre point de vue, l'inconvénient de ce travail réside dans leur processus de publication. Lors de cette étape, les fournisseurs doivent associer à leurs services une méthode de résolution de problème. Or, nous pensons que les services Web doivent être publiés dans un registre indépendamment d'une résolution particulière afin d'être facilement réutilisés dans le plus grand nombre de cas possibles.

3.4 Conclusion

Dans ce chapitre, nous avons étudié un ensemble de travaux, tant industriels qu'académiques, qui œuvrent pour la mise à disposition des services Web.

Dans un premier temps, nous avons vu, qu'à l'inverse du processus de description, il n'existe pas de solution standard pour la publication, la recherche, et la sélection de services Web. À sa création, UDDI avait cette ambition. Cependant, depuis la suppression des registres publics de UDDI en 2006, nous pouvons dire que cet objectif n'a pas été atteint. Cet arrêt de la maintenance des registres publics a engendré une baisse du nombre de registres de services Web classiques disponibles sur le Web. En effet, si nous comparons les registres de services Web disponibles aujourd'hui et ceux présents en 2006 [Bachlechner *et al.,* 2006], nous remarquons que beaucoup d'entre eux ont disparu (sur dix registres étudiés par [Bachlechner *et al.,* 2006], à ce jour plus que quatre sont disponibles). Ceci est dû au fait que la plupart de ces registres référençaient les registres publics UDDI.

Dans un second temps, nous avons étudié les travaux spécifiques à la publication, la recherche et la sélection de services Web sémantiques. De nombreux travaux œuvrent dans ce domaine puisqu'une description (et, par conséquent, la publication) strictement syntaxique des services Web ne peut permettre une recherche et une sélection efficaces de ces services. Ces travaux reposent sur des langages et des concepts issus du domaine du Web sémantique ou de l'intelligence artificielle et ainsi mettent en œuvre des mécanismes facilitant les étapes de recherche et de sélection pour les clients.

Le deux premiers chapitres de l'état de l'art sont constitués d'étude de travaux portant sur des services Web élémentaires. Or, afin d'implémenter un système à base d'AOS, les concepteurs ont besoin d'organiser un ensemble de services. Ceci engendre de nouvelles problématiques propres à la composition de services Web, sujet du prochain chapitre.

4 COMPOSITION DE SERVICES WEB

Dans les chapitres précédents, nous avons étudié la description, la publication, la recherche et la sélection de services Web élémentaires. Si l'objectif du concepteur d'une application n'est pas atteint par l'invocation d'un simple service Web élémentaire, alors le concepteur doit combiner les fonctionnalités d'un ensemble de services. Ce processus est appelé *composition de services Web* [Benatallah *et al.,* 2005] [Alonso *et al.*, 2004]. Les services Web invoqués lors d'une composition de services Web sont appelés *services Web composants*. La mise en œuvre d'une composition de services Web engendre des problèmes tels que la sélection des *services Web composants* et l'implémentation des interactions entre ces services.

Dans ce chapitre, nous présentons dans un premier temps les définitions et les types de composition de services Web présents dans la littérature. Ensuite, nous étudions quatre langages de composition de services Web. Enfin, un ensemble de travaux qui proposent des plates-formes d'exécution de la composition est décrit.

L'exemple qui sert d'illustration, dans les sections dédiées à l'étude des langages et des plates-formes de composition de services Web, est la composition de deux services Web nommés *GeoIPService* et *GlobalWeather*[45]. Cette composition de services Web, nommée *myMeteo*, est un service de météorologie. Le premier service (*GeoIPService*) permet de connaître la localisation de l'utilisateur, tandis que le second (*GlobalWeather*), à partir de la localisation, retourne les conditions météorologiques courantes.

4.1 Définitions et types de composition de services Web

Cette section a pour but d'exposer, d'une part, l'ensemble des définitions et objectifs de la composition de services selon différents points de vue rencontrés dans la littérature, et, d'autre part, les différents types de composition.

[45] Le service Web *GlobalWeather* est le service d'illustration dans le chapitre 1 consacré à l'étude de la description de service Web.

4.1.1 Définitions

Nous donnons tout d'abord une définition générale de la composition de services Web, puis abordons les différences d'interprétation de cette définition selon deux approches phares de la composition (le *e-business* et le Web sémantique).

Suite à l'étude de différents travaux sur la composition de services Web ([Alonso *et al.*, 2004], [Benatallah *et al.,* 2005], [Claro *et al.*, 2006], [Srivastava *et al.*, 2003], [Yang *et al.*, 2004]) nous retenons la définition de [Benatallah *et al.,* 2005] qui nous paraît la plus générale. Dans [Benatallah *et al.,* 2005], les auteurs considèrent la composition de services Web comme étant un moyen efficace pour créer, exécuter, et maintenir des services qui dépendent d'autres services. Ces mêmes auteurs ont défini le cycle de vie d'une composition de services Web reposant à partir de six activités [Benatallah *et al.*, 2002] (*cf.* Figure 4.1) :

L'encapsulation de services natifs (*Wrapping services*). Cette première activité permet de s'assurer que tout service peut être appelé lors d'une composition, indépendamment de son modèle de données, de son format de message, et de son protocole d'interaction.

L'établissement d'accord d'externalisation (*Setting outsourcing agreements*). Cette seconde activité consiste à négocier, établir, et appliquer des obligations contractuelles entre les services.

L'assemblage de services composants (*Assembling composite services*). Cette activité permet de spécifier, à un haut niveau d'abstraction, l'ensemble des services à composer afin d'atteindre l'objectif attendu. Cet assemblage comporte une phase d'identification des services et de spécification de leurs interactions conformément aux descriptions et aux accords entre services.

L'exécution de services composants (*Executing services*). Cette activité consiste en l'exécution des spécifications de la composition précédemment définies.

Le contrôle de l'exécution de services composites (*Monitoring services*). La phase de contrôle permet de superviser l'exécution de la composition en vérifiant, par exemple, l'accès aux services, les changements de statut, les échanges de messages. Ce contrôle permet de détecter des violations de contrats, de mesurer les performances des services appelés et de prédire des exceptions.

L'évolutivité des services (*Evolving services*). Cette dernière phase permet de faire évoluer la composition en modifiant les altérations de l'organisation de services, en utilisant de nouveaux services, ou en prenant en compte les retours de la phase de contrôle.

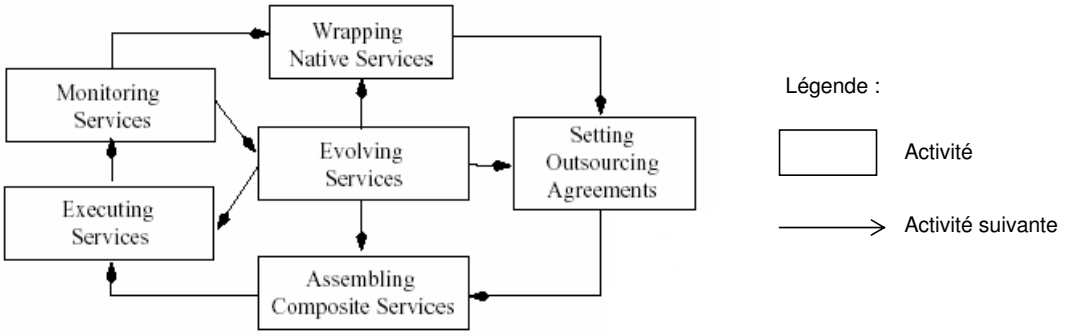


Figure 4.1 Illustration du cycle de vie de d'une composition de services Web par [Benatallah *et al.*, 2002].

Nous considérons qu'il existe deux domaines d'application de la composition de services Web (le *e-business* et le Web sémantique). Cependant, chacun d'eux utilise la composition à des fins qui lui sont propres :

L'*e-business*. Les organisations utilisent des méthodes afin de représenter les flots de données inter et intra entreprises. Les concepteurs des applications d'entreprise (ou les experts métiers) définissent au préalable les processus. La définition du processus repose sur la spécification du besoin, des flots de contrôle entre les activités, et des contraintes. La composition de services Web permet d'attribuer à chaque activité du processus un service Web.

Le Web sémantique. Les travaux dans le domaine du Web sémantique rendent le Web exploitable par les machines elles-mêmes, sans intervention humaine. Certains travaux de ce domaine étendent ce même objectif (l'exploitation par les machines elles-mêmes) aux services Web. On parle alors de services Web sémantiques (*cf.* section 2.2.1). Dans le contexte de la composition de services Web, le Web sémantique intègre l'automatisation aux processus de découverte, d'invocation, et de surveillance des services Web [Payne *et al.,* 2004].

4.1.2 Types de composition de services Web

La plupart des travaux portant sur la composition de services Web reconnaissent deux types de composition : l'*orchestration* et la *chorégraphie* de services. Cependant, selon les travaux, les définitions des types de composition diffèrent. Pour [Peltz, 2003] et [Benatallah *et al.,* 2005], l'orchestration et la chorégraphie sont des moyens de concevoir la composition, tandis que dans [Barros *et al.*, 2005b], l'orchestration et la chorégraphie sont des points de vue de la composition de services. Dans notre travail, nous retenons le fait que l'orchestration et la chorégraphie sont des moyens de concevoir la composition et les désignons comme des types de composition.

Afin de choisir l'un ou l'autre de ces types de composition (orchestration ou chorégraphie), le concepteur de systèmes doit prendre en compte différents paramètres. Les paragraphes ci-dessous, à partir des définitions données par les travaux portant sur la composition de services Web, décrivent chacun de ces types.

4.1.2.1 Orchestration

[Barros *et al.*, 2005b] définissent l'orchestration comme un ensemble de processus exécutés dans un ordre prédéfini afin de répondre à un but. Ce type de composition permet de centraliser l'invocation des services Web composants. Chaque service est décrit en termes d'actions internes. Les contrats entre deux services sont constitués selon le processus à exécuter.

À l'instar de [Barros *et al.*, 2005b], [Benatallah *et al.,* 2005] définissent l'orchestration comme un processus exécutable. [Benatallah *et al.,* 2005] ajoutent que l'orchestration est un ensemble d'actions à réaliser par l'intermédiaire de services Web. Un moteur d'exécution, un service Web jouant le rôle de chef d'orchestre, gère l'enchaînement des services Web par une logique de contrôle. Pour concevoir une orchestration de services Web, il faut décrire les interactions entre le moteur d'exécution et les services Web. Ces interactions correspondent aux appels, effectués par le moteur, d'action(s) proposée(s) par les *services Web composants*.

D'après [Peltz, 2003], l'orchestration de services Web consiste en la programmation d'un moteur qui appelle un ensemble de services Web selon un processus prédéfini. Ce moteur définit le processus dans son ensemble et appelle les services Web (tant internes qu'externes à l'organisation) selon l'ordre des tâches d'exécution. La Figure 4.2 illustre l'exécution du moteur (lui-même un service Web – *Service Web Moteur*) permise par l'enchaînement de l'exécution de deux autres services Web (le *Service Web 1* puis le *Service Web 2*). Cet enchaînement est possible via un opérateur d'ordonnancement (représenté par le losange dans la figure). L'exécution de la composition repose sur l'appel du *Service Web 1*, puis sur l'appel du *Service Web 2*, réalisés tous deux par le *Service Web Moteur*.

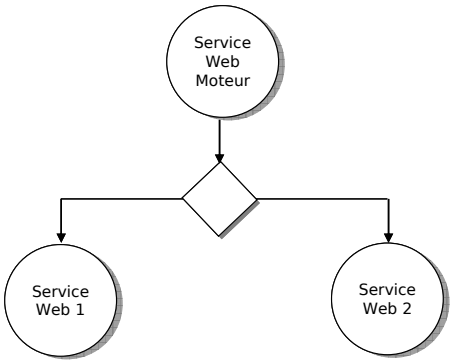


Figure 4.2 Illustration de l'orchestration, d'après [Peltz, 2003].

L'orchestration peut être vue comme une composition ascendante : les services Web utilisés dans la composition existent au préalable et sont appelés selon un enchaînement prédéfini afin de réaliser un processus précis. La Figure 4.3 illustre l'orchestration. La requête du client (logiciel ou humain) est transmise au moteur d'exécution (*Moteur*). Ce dernier, d'après le processus préalablement défini, appelle les services Web (ici, *SW1*, *SW2*, *SW3* et *SW4*) selon l'ordre d'exécution.

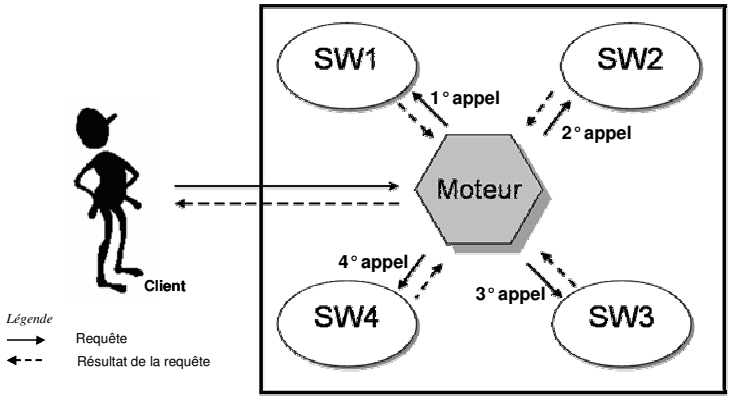


Figure 4.3 Vue générale de l'orchestration.

En d'autres termes, l'orchestration de services Web exige de définir l'enchaînement des services Web selon un canevas prédéfini, et de les exécuter selon un script d'orchestration. Ces derniers (le canevas et le script) décrivent les interactions entre services Web en identifiant les messages, et en spécifiant la logique et les séquences d'invocation. Le module exécutant le script d'orchestration de services Web est appelé un moteur d'orchestration. Ce moteur d'orchestration est une entité logicielle qui joue le rôle d'intermédiaire entre les services en les appelant suivant le script d'orchestration.

4.1.2.2 Chorégraphie

D'après [Barros *et al.*, 2005b], la chorégraphie permet de décrire la composition comme un moyen d'atteindre un but commun en utilisant un ensemble de services Web. La collaboration entre chaque service Web de la collection (faisant partie de la composition) est décrite par des flots de contrôle. Le fait que la chorégraphie mette en œuvre un ensemble de services Web afin d'accomplir un but commun apparaît aussi dans les travaux de [Benatallah *et al.,* 2005]. Pour concevoir une chorégraphie, les interactions entre les différents services doivent être décrites. La logique de contrôle est supervisée par chacun des services intervenant dans la composition. L'exécution du processus est alors distribuée.

D'après [Peltz, 2003], la description de chaque service Web intervenant dans la chorégraphie inclut la description de sa participation dans le processus. De ce fait, ces services peuvent collaborer à l'aide de messages échangés afin de savoir si tel ou tel service peut aider dans l'exécution de la requête. Chaque service Web peut communiquer avec un autre service Web par l'intermédiaire d'échange de

messages. La Figure 4.4 représente un protocole d'initiation de collaboration entre deux services dans le cadre d'une chorégraphie. Dans cet exemple, le *Service Web 1* demande l'exécution d'une méthode du *Service Web 2* par l'intermédiaire d'un envoi de message (*Requête de service*). Cette requête est acceptée par le service Web 2. Ce dernier envoi un message d'acceptation au *Service Web 1* (*Acceptation*). Le Service Web 1 accepte le service proposé par le Service Web 2 en lui envoyant un message (*Service admis*) accordé (*Acceptation*) par ce second service. Une fois ces messages échangés le *Service Web 1* peut invoquer les *Service Web 2* dans le cadre de la composition.

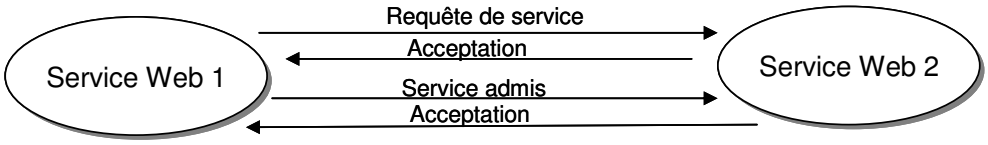


Figure 4.4 L'illustration de la chorégraphie, d'après [Peltz, 2003].

La chorégraphie est aussi appelée composition dynamique [Peltz, 2003]. En effet, l'exécution n'est pas régie de manière statique comme dans une composition de type orchestration. Dans une chorégraphie, à chaque pas de l'exécution (*i.e.* à chaque étape de la composition), un service Web choisit le service Web qui lui succède et implémente ainsi une partie de la chorégraphie. La composition de type chorégraphie n'est pas connue, ni décrite à l'avance.

Le W3C compte depuis 2002 parmi ses groupes de travail, le groupe de travail sur la chorégraphie de services Web (*Web Services Choreography Working Group*[46]). Pour ce dernier, la chorégraphie des services Web concerne les interactions observables des services avec leurs utilisateurs (appelés aussi clients) [Austin *et al.,* 2004]. Ces utilisateurs, automatisés ou non, peuvent être d'autres services Web, des applications, ou des concepteurs d'applications. Cet ensemble spécifique d'interactions peut être comparé à une collaboration entre un ensemble de services Web et leurs clients. La description d'une chorégraphie est un contrat multi-parties qui décrit, à partir d'un point de vue global, le comportement observable externe entre plusieurs clients (généralement des services Web). Chaque comportement externe observable est défini comme la présence ou l'absence de messages échangés entre un service Web et ses clients.

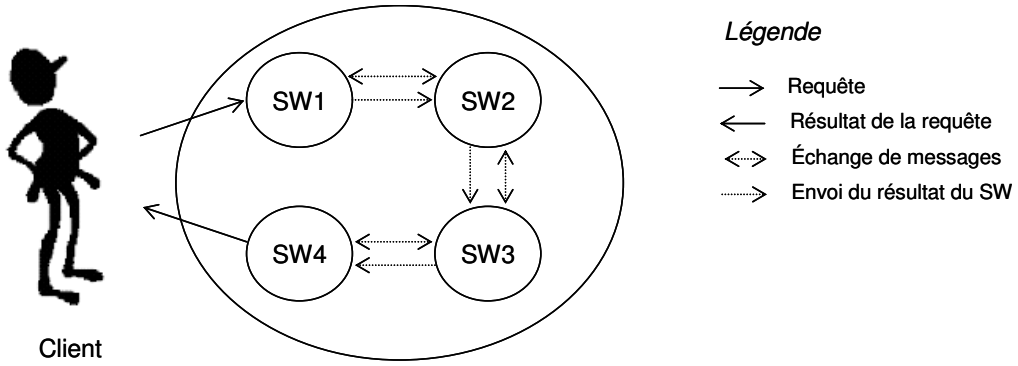


Figure 4.5 Vue générale de l'exécution d'une composition de services Web de type chorégraphie.

La Figure 4.5 permet d'illustrer une vue générale d'une composition de services Web de type chorégraphie. Le client (logiciel ou humain) établit une requête qui est satisfaite par l'exécution automatique de quatre services Web (SW1, SW2, SW3 et SW4). La requête de l'utilisateur est transmise au premier service Web (SW1) qui est exécuté. Le SW1 découvre ensuite le service Web lui succédant. Le processus de découverte repose, selon les cas, soit sur une recherche dans un registre local ou public, soit sur une découverte globale sur le Web à l'aide d'ontologies. Une fois le service

[46] http://www.w3.org/2002/ws/chor/

découvert, les deux services (SW1 et SW2) échangent des messages (comme illustré par la Figure 4.4) afin de vérifier si leur communication est viable dans le cadre de la requête. Si les échanges de messages sont concluant, le résultat de l'action du SW1 est transmis au SW2 qui l'utilise comme paramètre d'entrée. Le processus d'implémentation de la composition est identique pour chaque étape (SW2 et SW3). Le SW4 termine le processus et le résultat de son action est transmis au client.

4.2 Langages de composition de services Web

De nombreux industriels (tels qu'IBM[47] ou Microsoft[48]) et consortium (tel que le W3C) travaillent afin de mettre en œuvre un langage de composition de services Web standard (tel que WSCI – *Web Service Choreography Integration* [Arkin *et al.,* 2002]). Dans cette section, nous étudions les langages qui sont soit largement utilisés dans l'industrie (BPEL4WS [Andrews *et al.,* 2003]), soit en cours de standardisation (WS-CDL [Kavantzas *et al.,* 2005] et OWL-S [Martin *et al.*, 2004]). Pour chaque langage étudié, nous indiquons l'origine du travail, les bases théoriques sur lesquelles le langage est construit et une illustration de la description d'une composition de services.

4.2.1 BPEL4WS

BEA[49], IBM, SAP[50], Siebel Systems[51] et Microsoft ont uni leurs efforts afin de produire un langage de composition de services Web, conçu pour supporter les processus métier à travers les services Web. Ce langage, BPEL4WS (*Business Process Execution Language for Web Services*), est issu de la fusion de deux langages : WSFL – *Web Service Flow Language* [Leymann, 2001] d'IBM et XLANG [Thatte, 2001] de Microsoft. BPEL4WS combine les caractéristiques d'un langage de processus structuré par bloc (XLANG) avec ceux d'un langage de processus basé sur les processus métier (WSFL).

BPEL4WS [Andrews *et al.,* 2003] est basé sur XML et sur les *workflows*. Ce langage distingue les processus abstraits des processus exécutables.

Le processus abstrait. Ce processus spécifie les messages échangés entre les différentes parties (*services Web composants*) sans indiquer le comportement de chacune d'elles. [Wynen, 2003] parle de *Business Protocol*, c'est-à-dire la spécification du comportement des partenaires par rapport aux messages échangés, sans rendre public le comportement interne. Ce processus abstrait peut être relié à une composition de type chorégraphie. Les services Web communiquent alors à l'aide d'échanges de messages (*cf.* partie gauche de la Figure 4.6).

Le processus exécutable. Ce processus permet de spécifier l'ordre d'exécution des activités, le partenaire concerné, les messages échangés entre ces partenaires, et les mécanismes des erreurs et des exceptions. En d'autres termes, il s'agit du moteur de l'orchestration donnant une représentation indépendante des interactions entre les partenaires.

La Figure 4.6 illustre la mise en œuvre des deux types de processus (exécutable et abstrait) par BPEL4WS. La définition du processus métier est donnée par le processus exécutable. Les processus abstraits gèrent les invocations entre les différents services Web (*WS*) permettant l'exécution de la composition de services définie dans le processus exécutable.

[47] http://www.ibm.com

[48] http://research.microsoft.com

[49] http://fr.bea.com/

[50] http://www.sap.com/

[51] http://www.siebel.com/

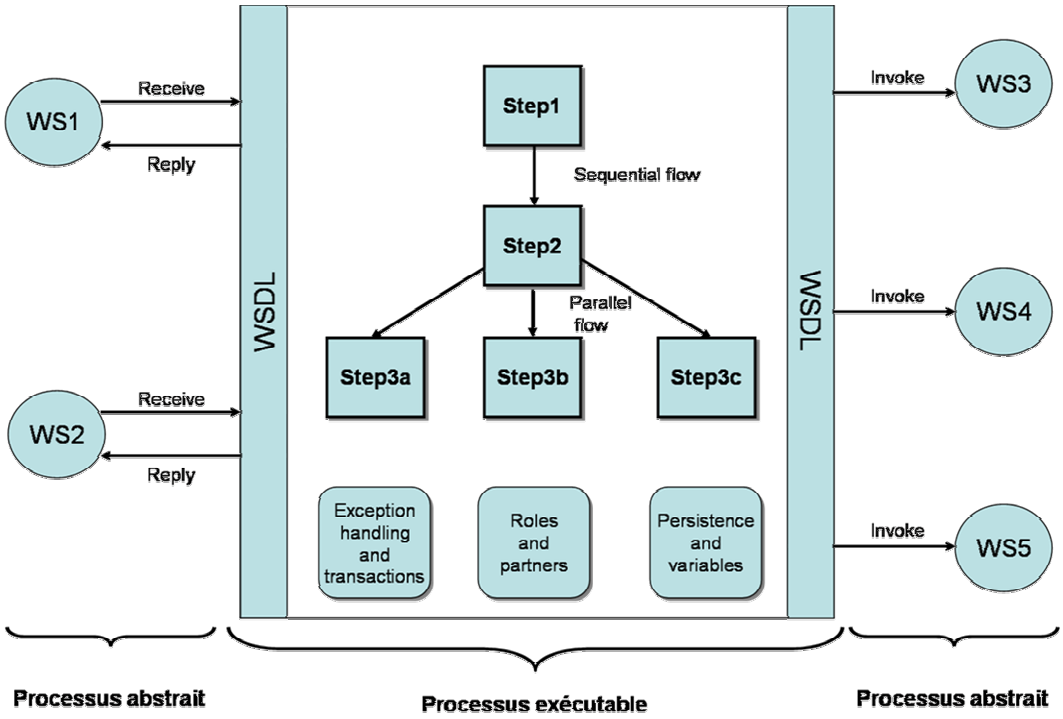


Figure 4.6 Le flot de processus avec BPEL4WS, d'après [Peltz, 2003].

Trois éléments permettent à BPEL4WS de gérer le flot de processus dans le processus exécutable : les transactions (*Exception handling and transactions*), les partenaires (*Roles and Partners*) et les espaces de stockage (*Persistance and Containers*) (*cf.* Figure 4.6).

Les transactions. Les transactions sont utilisées dans BPEL4WS pour gérer les erreurs et les appels d'autres services si le service appelé est indisponible ou défaillant.

Les partenaires. Les partenaires sont différents services Web invoqués dans le processus. Ils ont chacun un rôle spécifique dans un processus donné. Chaque partenaire est décrit par son nom, son rôle (en tant que service indépendant), et son rôle dans le processus. Dans la Figure 4.7, présentant la définition d'un processus exécutable en BPEL4WS, nous pouvons voir les définitions de deux partenaires (des lignes 2 à 11). Le fait de décrire deux niveaux de rôle permet à chaque partenaire d'avoir une vie indépendante des compositions dans lesquelles il intervient.

Les espaces de stockage. Les espaces de stockage permettent la transmission des données. Le flot de processus BPEL4WS permet que ces données soient cohérentes à travers les messages échangés entre les services Web. Un message peut être un message d'appel (*invoke*), de réponse (*reply*) ou d'attente (*receive*).

La structure des différentes activités peut être, soit séquentielle (*Sequential flow*), soit parallèle (*Parallel flow*). Si l'activité est parallèle, alors plusieurs services peuvent être invoqués en même temps.

À partir de la Figure 4.7, nous pouvons mettre en relief la syntaxe de BPEL4WS en décrivant le processus `myMeteo` (élément `process`, élément racine du document BPEL4WS, ligne 1). Les différents partenaires (`geoIPService` et `globalWeather`) sont en premier lieu définis (élément `partners`, lignes 2 à 11). Ces deux acteurs interagissent avec le service qui orchestre la composition par l'intermédiaire de deux messages nommés `getLocalisation` et `getWeather` (élément `containers`, lignes 12 à 15). L'élément `containers` référence le type de message (attribut `messageType`, lignes 13 et 14) contenus dans les descriptions WSDL des *services Web composants*. Ainsi, le service Web qui orchestre connaît les types de données manipulées. Le flot de processus

(élément `flow`, lignes 16 à 25) se compose tout d'abord de l'appel du partenaire `geoIPService` (élément `invoke`, lignes 17 à 20) puis de l'appel du partenaire `globalWeather` (élément `invoke`, lignes 21 à 24). L'élément `invoke` comprend les accès aux ports (attribut `portType`, lignes 18 et 22) et les opérations (attribut `operation`, lignes 19 et 23) référençant la description WSDL des *services Web composants*. La Figure 4.7 n'illustre qu'un extrait de la description du processus exécutable *myMeteo*. BPEL4WS intègre aussi les flots de données dans la description du processus en reliant les sorties d'un service avec les entrées du service lui succédant.

```
1  <process name="myMeteo">
2      <partners>
3          <partner name="geoIPService"
4                   serviceLinkType="geoIPLink"
5                   myRole="geoIP"
6                   partneRole="geoIPContext"/>
7          <partner name="globalWeather"
8                   serviceLinkType="globalWeatherLink"
9                   myRole="weather"
10                  partnerRole="globalWeatherService"/>
11     </partners>
12     <containers>
13         <container name="getLocalisation" messageType="getGeoIPContextSoapIn"/>
14         <container name="getWeatherContainer" messageType="getWeatherSoapIn"/>
15     </containers>
16     <flow>
17         <invoke partner="geoIPService"
18                 portType="GeoIPServiceSoap"
19                 operation="GetGeoIPContext"
20                 inputContainer="getLocalisation" />
21         <invoke partner="globalWeather"
22                 portType="GlobalWeatherSoap"
23                 operation="GetWeather"
24                 inputContainer="getWeatherContainer" />
25     </flow>
26 </process>
```

Figure 4.7 Définition du processus exécutable de la demande du service de météorologie *myMeteo* à l'aide de BPEL4WS.

BPEL4WS est le premier langage de composition de services Web adopté par la communauté des services Web. Ceci est principalement dû au fait que ce langage possède une grande expressivité dans la définition du processus exécutable [Wohed *et al.*, 2003]. L'inconvénient principal de BPEL4WS est que la définition du processus est rigide. Si une activité du processus échoue, le processus dans son intégralité échoue. Aucun retour en arrière et aucune alternative au processus ne sont possibles. De même, lors de la description du processus exécutable, la définition des flots de données ne permet pas de connecter des services dont les entrées/sorties ne correspondent pas exactement. BPEL4WS ne prévoit pas de mécanisme de transformation de données.

4.2.2 WS-CDL

WS-CDL (*Web Service Choreography Description Language*) [Kavantzas *et al.,* 2005] est un langage issu des efforts de standardisation du groupe de travail du W3C portant sur la chorégraphie de services Web (*Web Services Choreography Working Group*[52]). L'objectif de ce langage est de décrire les relations entre les services Web lors d'une composition de type chorégraphie.

[52] http://www.w3.org/2002/ws/chor/

WS-CDL, à l'instar des standards de services Web, est basé sur XML. Il complète la description WSDL des services Web afin de décrire les interactions entre les participants (les autres services) de la composition. L'élément `Package` englobe cette description (*cf.* Figure 4.8). La description des interactions du service étant complexe, nous décomposons ici l'exemple de syntaxe en illustrant les éléments que nous jugeons importants.

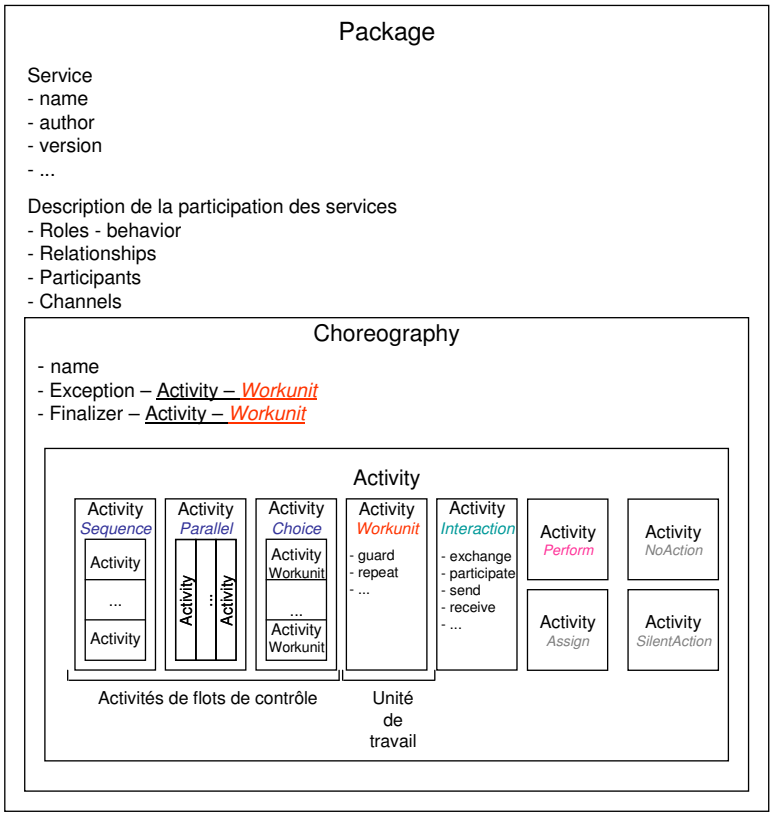


Figure 4.8 Représentation du package de WS-CDL, d'après [Barros *et al.*, 2005a].

Le `Package` est composé de trois parties : la description du service et des participants à la composition (premiers éléments du `Package`), la description de la chorégraphie (élément `Choreography`) et la description des activités (élément `Activity`).

Les premières informations décrites dans WS-CDL sont relatives à la description du service Web : nom du service (élément `name`), du fournisseur (élément `author`), de la version du service (élément `version`). La suite de la description décrit plus particulièrement la participation du service au sein de la composition (éléments `Roles`, `Behavior`, `Relationships`, `Participants` et `Channels`).

```
1  <roleType name="geoIP">
2      <behavior name="getLocalisation"
3                interface="http://www.webservicex.com/geoipservice.asmx?wsdl"/>
4  </roleType>
```

Figure 4.9 Définition du rôle et du comportement du service *GeoIPService*.

Rôles et comportement (`Roles` et `Behavior`). Un rôle (élément `roleType`) décrit les comportements (sous-élément `behavior`) observables d'un service lors des interactions entre services (ou participants). Un comportement peut référencer une interface (attribut `interface`). Cette interface, si elle est décrite, est le fichier WSDL de description du participant. Dans la Figure 4.9, nous décrivons le rôle (valeur `geoIP` de l'attribut `name` de l'élément `roleType`, *cf.* ligne 1) du service *GeoIPService* et son comportement (valeur `getLocalisation` de l'attribut `name` de l'élément `behavior`, ligne 2). Le comportement référence le fichier de description

WSDL (valeur de l'attribut `interface` de l'élément `behavior`, ligne 3), description appelée, dans WS-CDL, *interface du comportement*.

Relations (`Relationships`). Une relation identifie le lien entre les rôles. Ce lien permet de mettre en œuvre la collaboration entre les services. Dans notre exemple, le rôle (`geoIP`) du service *GeoIPService* et le rôle (`weather`) du service *GlobalWeather* collaborent afin de retourner au client les conditions météorologiques sans lui demander au préalable sa localisation. La Figure 4.10 illustre la description des deux rôles (`geoIP` et `weather`) nécessaires à la composition `myMeteo`.

```
1  <relationshipType name="myMeteo">
2      <role type="geoIP"/>
3      <role type="weather"/>
4  </relationshipType>
```

Figure 4.10 Définition de la relation entre les services Web *GeoIPService* et *GlobalWeather*.

Participants (`Participants`). Cet élément permet de regrouper l'ensemble des rôles joués par un même participant. Chaque service (participant) mis en jeu dans la composition de services, voit ses rôles énumérés dans l'élément `Participant`. Il existe autant d'élément `Participant` que de service dans la composition. Dans notre exemple, chaque participant ne joue qu'un rôle.

Canaux (`Channels`). Un canal spécifie où et comment les informations sont échangées lors d'une interaction entre participants. La description d'un canal comprend la description du rôle et du comportement intervenant dans l'interaction.

La seconde partie de WS-CDL est la description de la chorégraphie (élément `Choreography`). Cet élément est réutilisable et définit les échanges de messages (*i.e.* la chorégraphie), les exceptions (élément `Exception`) et la finalisation de la chorégraphie (élément `Finalizer`). La description des échanges de messages est permise grâce à l'élément `Activity`, sous-élément de l'élément `Choreography`.

Les activités décrivent les actions mises en jeu dans la chorégraphie. Le langage WS-CDL comprend trois types d'activités : les activités de flots de contrôle, les activités d'unité de travail, et les activités basiques.

Activités de flots de contrôle. Dans WS-CDL, il existe trois activités de flots de contrôle : l'activité de séquence (élément `Sequence`), l'activité parallèle (élément `Parallel`) et l'activité de choix (élément `Choice`). Ces activités déterminent la manière dont les activités encapsulées s'exécutent. L'activité de séquence permet l'exécution en ordre séquentiel de l'ensemble des activités décrites. Les activités parallèles sont exécutées en même temps. L'activité de choix décrit l'exécution d'une activité choisie parmi un ensemble d'activités. Cet ensemble d'activités représente des choix.

Activités d'unité de travail (*Work Unit*). Le second type d'activité permise par WS-CDL est l'unité de travail. Cet élément décrit les pré-conditions (élément `guard`) et les éventuelles répétitions (élément `repeat`) d'une activité.

Activités basiques. Le langage WS-CDL propose de décrire quatre types d'activités basiques : les activités sans actions (éléments `NoAction` et `SilentAction`), les transferts de variables à l'intérieur d'un même rôle (élément `Assign`), les appels à d'autres chorégraphies (élément `Perform`) et les interactions entre participants (élément `Interaction`). Cette dernière activité décrit un échange d'informations. Une interaction peut établir trois actions (élément `action`) : une requête (élément `request`), une réponse (élément `respond`), et une requête qui nécessite une réponse (élément `request-respond`). Pour décrire une interaction il faut intégrer dans la

description de cette interaction la description des participants (élément `participate`), des informations échangées (élément `exchange`) et du canal d'échange d'informations (éléments `send` et `receive`).

Dans notre exemple de service de météorologie (*cf.* Figure 4.11), les activités sont des activités de séquence (élément `sequence`, lignes 5 à 15). Cette séquence est composée d'une interaction (élément `interaction`, lignes 6 à 14) nommé `getWeather` entre les services Web *GeoIPService* et *GlobalWeather*. Cette interaction s'effectue depuis le partenaire ayant pour rôle `geoIP` vers le partenaire ayant pour rôle `weather` (élément `participate`, lignes 7 et 8). L'interaction entre les deux services est une action de type `request` (valeur de l'attribut `action` de l'élément `exchange`, lignes 9 et 10). Le service Web *GeoIPService* invoque le service Web *GlobalWeather* en lui transmettant une localisation, qui est le résultat de sa propre exécution (ligne 12).

```
1  <package>
2      (...)
3      <choreography name="myMeteo" root="true">
4          <relationship type="myMeteo"/>
5          <sequence>
6              <interaction name="getweather">
7                  <participate relationshipType="myMeteo"
8                               fromRole="geoIP" toRole="weather"/>
9                  <exchange name="getweatherExchange"
10                           action="request"
11                           informationType="getWeather">
12                     <send variable="GetGeoIPContextResponse"/>
13                 </exchange>
14             </interaction>
15         </sequence>
16     </choreography>
17 </package>
```

Figure 4.11 Extrait de l'interaction entre les deux services Web *GeoIPService* et *GlobalWeather* pour implémenter la composition *myMeteo*.

WS-CDL est un langage de composition de services de type chorégraphie définissant des contrats multi-parties. L'avantage de ce langage est que la description des interactions (l'élément `Choreography` du `Package`) est réutilisable. Ceci permet de diminuer la charge de travail des concepteurs. L'inconvénient de WS-CDL est que même si l'élément `Choreography` est réutilisable, sa description reste lourde. Un service doit posséder autant de `Package` que de participations à des compositions. De plus, WS-CDL n'inclut pas pour le moment de sémantique. Des travaux, tels que [Kang *et al.*, 2007a] et [Kang *et al.*, 2007b], proposent d'étendre WS-CDL afin d'y intégrer de nouveaux concepts (tels que la gestion des erreurs et des exceptions ou la mise en œuvre d'un minuteur d'exécution de la composition).

4.2.3 OWL-S

OWL-S [Martin *et al.*, 2004] est un langage de mise en œuvre de services Web sémantiques qui permet la description, la découverte, l'invocation et la composition de type chorégraphie des services Web. Dans la section 2.2.2.1 du chapitre 1, nous avons étudié OWL-S en tant que langage de description de services Web sémantiques élémentaires. Nous consacrons cette section à la description de leur composition.

Nous rappelons qu'une description OWL-S d'un service Web sémantique est composée de trois documents OWL : le profil de service (*Service Profile*), le modèle du service (*Service Model*) et l'accès au service (*Service Grounding*). Ces trois documents ont été étudiés dans la section 2.2.2.1 du chapitre 1. Dans cette section, nous revenons plus particulièrement sur le modèle du service (*Service Model*) étant donné que ce document intègre la description de la composition de services Web.

Le modèle du service présente le fonctionnement du service et définit les compositions de services Web (ici appelés processus – *Process*) dans lesquelles les services Web interviennent. Ces interventions sont décrites à l'aide du modèle de processus (*Process Model*). Le *Process Model* permet de décrire la tâche que propose le service Web au sein de la composition de services. Le modèle de processus est représenté par la classe *ProcessModel*. Cette dernière possède les propriétés suivantes : les paramètres d'entrée et de sortie, les participants au processus (les autres services Web intervenant dans la composition), les pré-conditions et les actions du service décrit. Il existe trois types de processus : le processus atomique, simple et composé (*cf.* Figure 4.12).

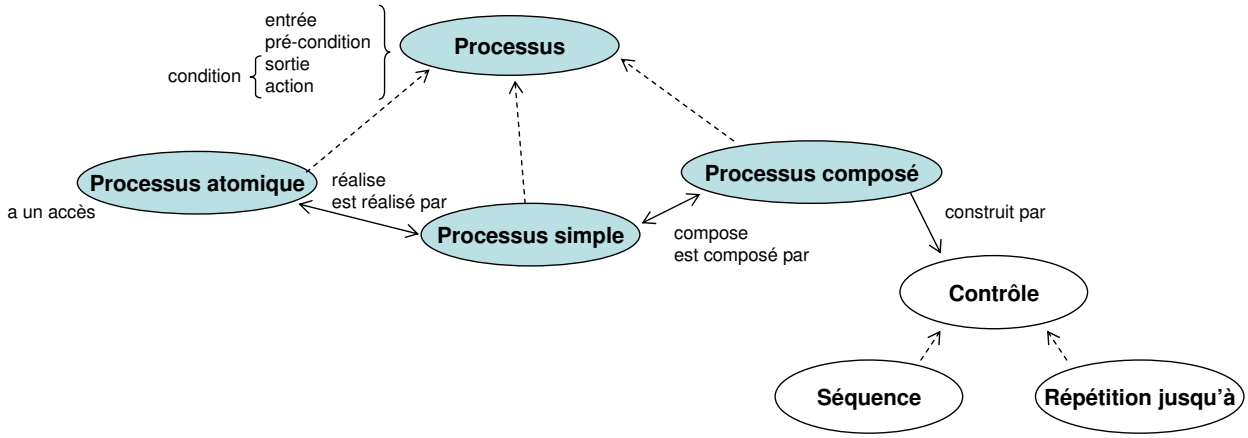


Figure 4.12 Description des différents types de processus, d'après [Martin *et al.*, 2004].

Le processus atomique (*Atomic Process*). Ce processus est directement invoqué par l'intermédiaire d'un accès (utilisation du *Grounding Profile* – *cf.* 2.2.2.1 du chapitre 1).

Le processus simple (*Simple Process*). Le processus simple n'est pas directement invoqué. Il fournit une vue d'un processus atomique ou la représentation simplifiée d'un processus composé.

Le processus composé (*Composite Process*). Le processus composant est décomposable en processus simples. Il utilise des commandes de contrôle (*Control Construct*) afin de gérer l'invocation des processus le composant. Dans OWL-S, il existe deux types de commande de contrôle : la séquence – *Sequence* (un service Web est appelé, puis un autre), et la répétition conditionnelle – *Repeat until* (un service Web est appelé jusqu'à atteindre la valeur d'une condition).

Les composants principaux du modèle de processus (*Process Model*) sont l'ontologie du processus (*Process Ontology*) et l'ontologie de contrôle du processus (*Process Control Ontology*).

L'ontologie du processus (*Process Ontology*). Cette ontologie décrit un service en termes de paramètres d'entrée et de sortie, de pré-conditions, des actions du service et dans le cas approprié, des sous-processus. Cette ontologie peut être utilisée afin de supporter l'invocation et la composition automatique de services Web.

L'ontologie de contrôle du processus (*Process Control Ontology*). Cette ontologie de contrôle du processus décrit chaque processus comme étant un état, en prenant en compte son activation, son exécution et sa terminaison.

```
1   <rdf:RDF (...)>
2   (...)
3       <process:ProcessModel rdf:ID="myMeteoProcessModel">
4           <process:hasProcess rdf:resource="#myMeteoProcess" />
5       </process:ProcessModel>
6   (...)
7       <process:CompositeProcess rdf:ID="myMeteo">
8           <process:composedOf>
9               <process:Sequence>
10                  <process:components rdf:parseType="Collection">
11                      <process:AtomicProcess rdf:about="#getGeoIP" />
12                      <process:AtomicProcess rdf:about="#getWeather" />
13                  </process:components>
14              </process:Sequence>
15          </process:composedOf>
16      </process:CompositeProcess>
17  </rdf:RDF>
```

Figure 4.13 Extrait du modèle de processus (*ProcessModel*) de la composition *myMeteo*.

La Figure 4.13 illustre la définition du modèle du service *GeoIPService* entrant dans la composition *myMeteo*. Le processus composé *myMeteo* (`<process:CompositeProcess rdf:ID="myMeteo">`, ligne 7) est composé de deux processus atomiques `getGeoIP` (`<process:AtomicProcess rdf:about="#getGeoIP"/>`, ligne 11) et `getWeather` (`<process:AtomicProcess rdf:about="#getWeather"/>`, ligne 12). L'interaction entre ces deux processus atomiques est séquentielle (élément `process:Sequence`, lignes 9 à 14).

OWL-S permet de décrire les compositions de type chorégraphie. Puisqu'il utilise les langages RDF et OWL, il permet une description sémantique des services Web intervenant dans la composition. L'inconvénient majeur de OWL-S est que la description d'un service Web est complexe étant donné qu'un service Web doit contenir autant de descriptions de processus que de compositions auxquelles il participe.

4.3 Plates-formes de composition de services Web

Dans cette section, nous étudions les travaux issus du monde académique qui proposent des plates-formes d'exécution de composition de services Web. Une plate-forme de composition de services Web répond aux problématiques de la composition de services Web non résolues par les langages de composition (telles que la mise en œuvre d'une composition dynamique, la compensation de services et le contrôle de l'exécution).

Pour chacune des quatre plates-formes étudiées (SELF-SERV, METEOR-S, SHOP2 et IRS-III), nous décrivons l'origine du travail, le modèle sous-jacent aux plates-formes et leur architecture. Les plates-formes industrielles (telles que Websphere[53] d'IBM, Weblogic[54] de BEA ou .NET[55] de Microsoft) ne sont pas présentées dans ce mémoire, étant donné qu'elles ne résolvent pas les problématiques de la composition de services Web auxquelles nous nous intéressons (mise en œuvre d'une composition dynamique, compensation de services et contrôle de l'exécution).

[53] http://www-306.ibm.com/software/websphere/

[54] http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/server

[55] http://www.microsoft.com/france/msdn/net/default.mspx

4.3.1 SELF-SERV

SELF-SERV (*compoSing wEb accessibLe inFormation and buSiness sERVice*) [Sheng *et al.*, 2002], [Benatallah *et al.,* 2003] est issu des travaux de deux universités australiennes[56]. Cette plate-forme compose les services Web dans un environnement pair à pair (*Peer to Peer*) [Shizuka *et al.*, 2004].

L'architecture de SELF-SERV est composée de deux grandes parties : le gestionnaire de services (*Service Manager*) et le *Pool de Services* (*cf.* Figure 4.14). Le gestionnaire de service joue le rôle de registre dans l'architecture classique des services Web et est fortement lié à UDDI (annuaire universel de services Web, *cf.* section 3.13 chapitre 3). La composition de services est gérée par le second composant de l'architecture (le *Pool de Services*).

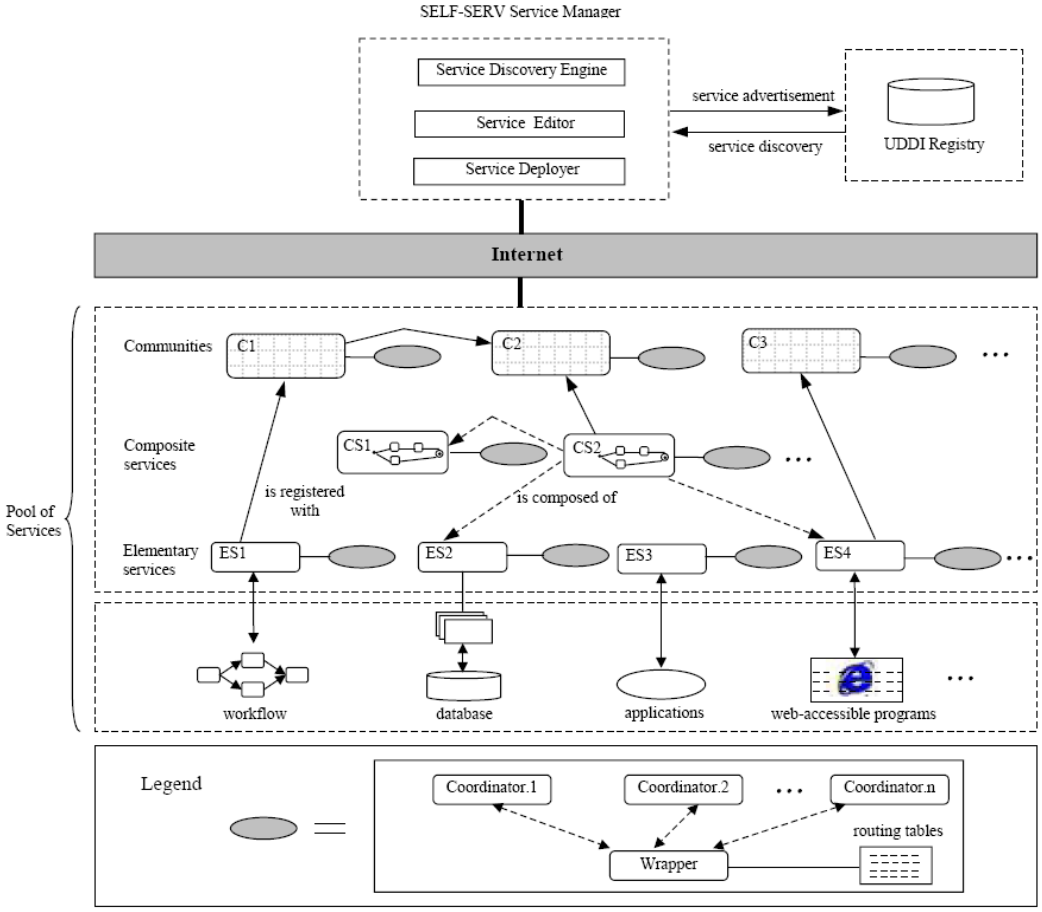


Figure 4.14 Architecture de SELF-SERV, d'après [Sheng *et al.*, 2002].

Dans le *Pool de services*, SELF-SERV distingue trois types de services (*cf.* Figure 4.14) :

Les services élémentaires (*elementary services*). Ces services sont des services Web élémentaires (ESA, ES2, ES3 et ES4).

Les services composés (*composite services*). Les services composés regroupent un ensemble de services. Chaque composition (CS1 et CS2) est décrite par un diagramme d'états UML. Chaque service est associé à un état du diagramme.

[56] L'Université de New South Wales, à Sidney (http://www.cse.unsw.edu.au/), et l'Université de Technologie de Brisbane (http://www.fit.qut.edu.au/).

Les communautés de services (*service communities*). Une communauté de services (C1, C2 et C3) rassemble un ensemble de services proposant une même activité (dans le cas de SELF-SERV, pouvant être mis en relation avec un même état d'un diagramme d'états).

Afin de mettre en œuvre la composition, SELF-SERV possède des composants logiciels (*peer*) appelés coordinateurs (*Coordinator*). Chaque état du service composé possède son propre coordinateur dont le rôle est d'appeler le service Web associé. Grâce à la technologie P2P, les coordinateurs communiquent entre eux afin d'échanger des données. Des tables de routage (*Routing Table*) gèrent la connaissance extraite des diagrammes d'états telle que le flot de contrôle. Cette connaissance permet la planification des coordinateurs.

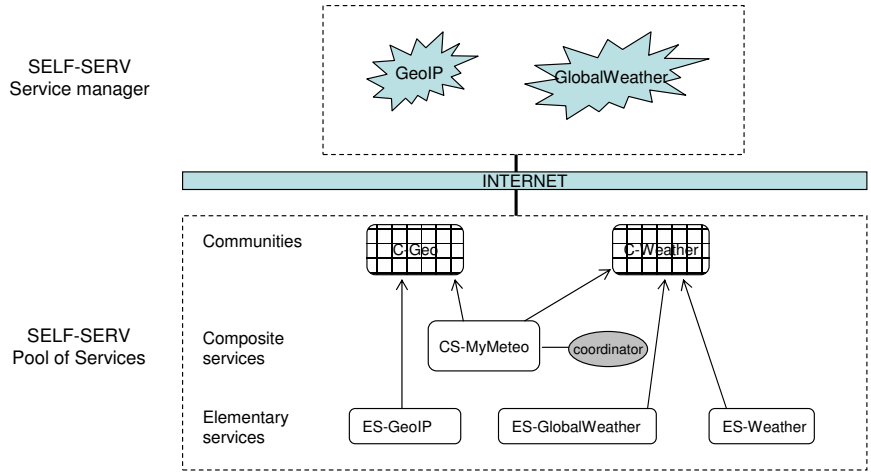


Figure 4.15 Illustration du fonctionnement de SELF-SERV à l'aide de l'exemple de la composition *MyMeteo*.

La Figure 4.15 illustre le fonctionnement de SELF-SERV à l'aide de l'exemple de composition de services Web *MyMeteo*. Cette composition est décrite par le bais d'un service composé (*CS-MyMeteo*) formalisé à l'aide d'un diagramme d'états UML. À chaque état est associé un coordinateur (*coordinator*) chargé de l'invocation proprement dite des services Web composants. Dans cet exemple, SELF-SERV possède deux communautés (*C-Geo* et *C-Weather*) auxquels sont associés des services élémentaires (respectivement, *ES-GeoIP* ainsi que *ES-GlobalWeather* et *ES-Weather*). Les coordinateurs sélectionnent le service approprié pour chaque communauté. Une fois les services choisis, ils sont invoqués via le gestionnaire de services (*Service Manager*).

Le premier avantage de SELF-SERV est que cette plate-forme possède un environnement graphique de définition du processus basé sur les diagrammes d'états. Ceci permet une utilisation aisée de la plate-forme. Le second avantage est le concept de communauté de services. Le fait d'avoir à disposition un ensemble de services répondant à une activité est important pour la sélection de services et la compensation de services. Ce concept de communauté a été intégré à des travaux de thèse portant sur la composition de services Web à l'aide de propriétés transactionnelles [Duarte-Amaya, 2007]. L'inconvénient de cette plate-forme est le fait que chaque état du diagramme d'état (*i.e.* chaque composant d'une composition) soit géré par un coordinateur (agent logiciel). Ceci rend l'exécution du processus plus lourde (car, pour chaque état exécuté, un agent et un service Web doivent être appelés).

4.3.2 METEOR-S

La plate-forme METEOR-S (*Managing End-To-End OpeRations-Semantics*) fait partie du projet METEOR-S[57] du laboratoire LSDIS[58] de l'Université de Georgia englobant les services Web sémantiques et l'exécution de processus.

La mise en œuvre de la composition de services Web avec la plate-forme METEOR-S [Oldham *et al.*, 2004] [Aggarwal *et al.*, 2004] repose sur quatre grandes phases (*cf.* Figure 4.16) : la conception du processus abstrait, la découverte des services, l'analyse de contraintes et l'exécution de la composition.

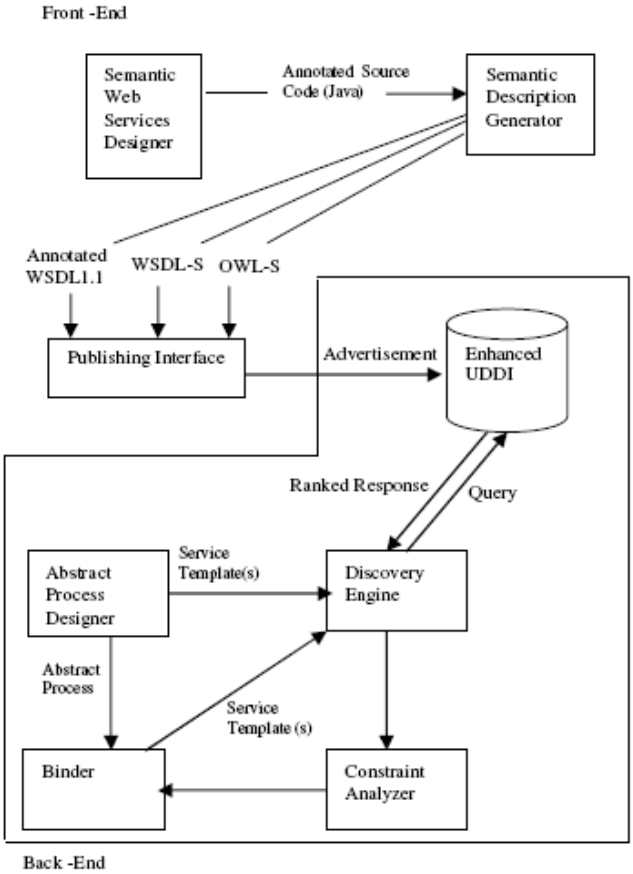


Figure 4.16 Architecture de METEOR-S, d'après [Aggarwal *et al.*, 2004].

Conception du processus abstrait (réalisée par le composant *Abstract Process Designer*). Afin de décrire les flots de contrôle entre les services, la plate-forme utilise le langage de composition BPEL4WS (*cf.* section 4.2.1). Pour chaque service, la plate-forme définit un *template* (S*ervice Template*). Ce patron contient les pré-conditions du services, les contraintes (en termes de qualité de service, de localisation de serveur, etc.) et la sémantique du service. Le composant *Abstract Process Designer* transmet, d'une part, les *Service Templates* au moteur de découverte (*Discovery Engine*) afin de sélectionner les services, et, d'autre part, le processus abstrait (*Abstract Process*) au moteur d'exécution (*Binder*) pour orchestrer la composition.

Découverte de services (réalisée par le composant *Discovery Engine*). Selon le *Service Template* le moteur de découverte (*Discovery Engine*) établit une requête au registre interne à l'architecture de METEOR-S (une version avancée de UDDI permettant de gérer de

[57] Projet *METEOR-S: Semantic Web Services and Processes* (http://lsdis.cs.uga.edu/projects/meteor-s/)

[58] LSDIS – *Large Scale Distributed Information Systems* (http://lsdis.cs.uga.edu/)

l'information sémantique à propos des services). Ce dernier retourne un ensemble de services correspondant au patron (*Template*) formalisé dans la requête (*query*).

Analyse de contraintes (réalisée par le composant *Constraint Analyser*). Le module *Constraint Analyser* filtre l'ensemble des services retournés par UDDI, selon la faisabilité du processus (par exemple, selon les convenances entre les paramètres de sortie d'un service et les paramètres d'entrée du suivant) et l'efficacité potentielle du processus (par exemple, selon la localisation du serveur hébergeant les services).

Exécution de la composition (réalisée par le composant *Binder*). Le module *Binder* met en œuvre l'exécution de la composition de services à l'aide, d'une part, de la description du processus abstrait établi en BPEL4WS, et, d'autre part, des services retenus par le module *Constraint Analyser*. Si le moteur d'exécution est confronté à un problème lors de l'exécution (tel qu'un service Web non disponible), le moteur transmet les *templates* des services posant problème au composant *Discovery Engine*. Les étapes de découverte des services et d'analyse de contraintes sont alors répétées.

La sémantique est aussi gérée à l'aide d'une extension de WSDL pour les services Web sémantiques (WSDL-S [Akkiraju *et al.,* 2005]) et du langage de description de services Web sémantiques (OWL-S). Le premier langage permet d'utiliser l'approche par annotation sémantique alors que le second utilise un ensemble d'ontologies. Les descriptions sémantiques des services sont enregistrées dans une version améliorée de UDDI (interne à METEOR-S).

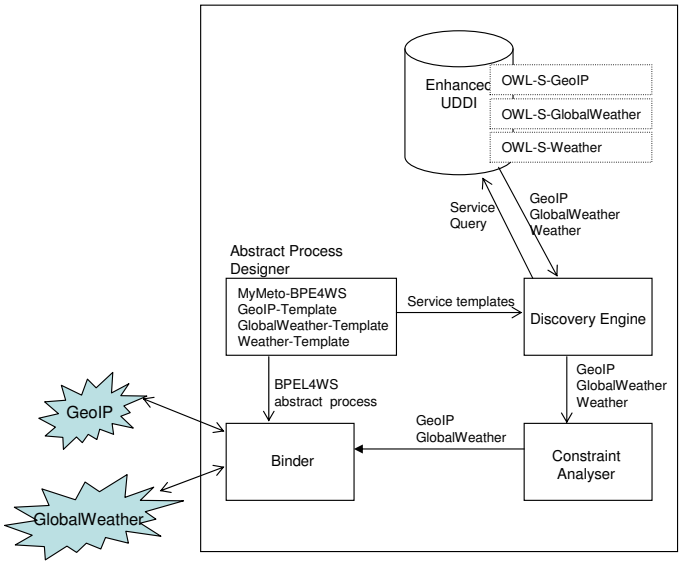


Figure 4.17 Illustration du fonctionnement de METEOR-S à l'aide de l'exemple de la composition *MyMeteo*.

Penons l'exemple de la composition de services Web *MyMeteo* pour illustrer le fonctionnement de METEOR-S (*cf.* Figure 4.17). Le module de conception de processus abstrait (*Abstract Process Designer*) contient la description de la composition *MyMeteo* (*MyMeteo-BPEL4WS*) formalisée à l'aide du langage BPEL4WS, ainsi que les *templates* des potentiels services composants (*GeoIP-Template*, *GlobalWeather-Template* et *Weather-Template*). Les *templates* sont transmis au module de découverte de services (*Discovery Engine*) afin que ce dernier recherche les services correspondant aux *templates* dans le registre (*Enhanced UDDI*). L'ensemble des services est retourné au module d'analyse de contraintes (*Constraint Analyser*) qui filtre les services. Dans notre exemple, il filtre le service *Weather*. Les services sélectionnés sont envoyés au module d'exécution (*Binder*) pour qu'il effectue l'invocation de ces derniers, selon le processus abstrait BPEL4WS (*BPEL4WS abstract process*), transmis par le module de conception de processus abstrait (*Abstract Process Designer*).

L'avantage de cette plate-forme est que le registre de services et la recherche sont basés sur la sémantique. De plus, le module d'analyse de contraintes permet de mettre en œuvre une forme basique d'adaptation en rapport avec la qualité de service et la localisation du service. L'inconvénient de cette plate-forme est que la description du processus repose entièrement sur BPEL4WS dont nous avons souligné la rigidité (*cf.* section 4.2.1).

4.3.3 SHOP2

La plate-forme SHOP2 appartient au projet SHOP[59] de l'Université de Maryland. L'objectif premier du projet SHOP est de proposer une plate-forme de planification hiérarchique [Erol *et al.*, 1994]. SHOP2 relie ce projet à la composition de services Web. D'autres travaux évoquent le fait d'utiliser la planification afin de composer les services Web, tels que [McDermott, 2002], [Srivastava *et al.,* 2003], ou encore [Carman *et al.,* 2003].

Le concept de base de SHOP2 [Sirin *et al*., 2004] repose sur le fait que cette plate-forme est basée sur les techniques du domaine de l'intelligence artificielle, et plus spécifiquement sur le concept de réseau de tâche hiérarchique (HTN – *Hierarchical Task Network*) [Nau *et al*., 2003]. Ce concept repose sur une vision hiérarchique du réseau de plans. Un réseau de plans est composé de tâches abstraites qui sont définies en termes d'actions primitives ou en termes d'autres tâches. Il existe trois types de tâches dans la planification HTN : les tâches de but, les tâches primitives et les tâches composées.

Les tâches de but. Ces tâches déterminent les propriétés pour atteindre l'état final.

Les tâches primitives. Ces tâches peuvent être directement invoquées.

Les tâches composées. Ces tâches dénotent les changements désirés en faisant référence à d'autres tâches (de but ou primitives).

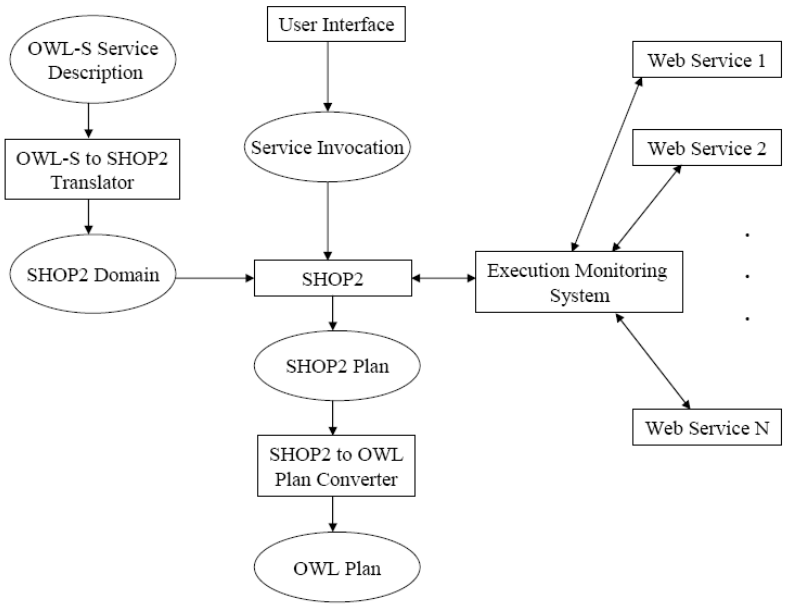


Figure 4.18 Architecture de SHOP2, d'après [Sirin *et al*., 2004].

La composition de services Web par l'intermédiaire de SHOP2 s'appuie sur deux phases : la définition de processus de composition et l'exécution des services.

[59] *Simple Hierarchical Ordered Planner* (http://www.cs.umd.edu/projects/shop/)

La définition du processus. La définition du processus peut se faire de deux manières. Soit l'utilisateur de la plate-forme définit le processus par l'intermédiaire d'un HTN, soit la description du processus existe déjà en OWL-S. Dans le second cas, le module *OWL-S to SHOP2 Translator* (*cf.* l'architecture de SHOP2 – Figure 4.18) se charge de la traduction en HTN. Le module *SHOP2 to OWL-S Plan Converter* convertit un processus défini dans la syntaxe de la plate-forme (HTN) en ontologies (au format OWL).

L'exécution du processus. Les services Web utilisés pour l'exécution de la composition doivent posséder une description OWL-S. À partir de la description du processus, l'*Execution Monitoring System* (*cf.* l'architecture de SHOP2 – Figure 4.18) fait appel aux services Web. Ce module garde en cache les résultats des services appelés pour les réutiliser si besoin est dans le processus. Ceci permet une plus grande rapidité de l'exécution de la composition.

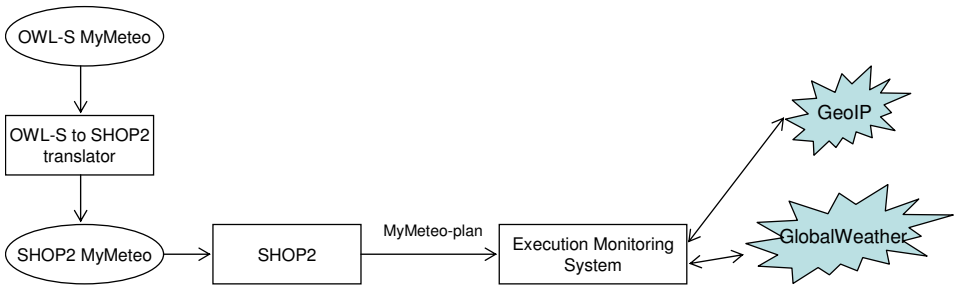


Figure 4.19 Illustration du fonctionnement de SHOP2 à l'aide de l'exemple de la composition *MyMeteo*.

Pour illustrer le fonctionnement de SHOP2 nous prenons l'exemple de la composition de services *MyMeteo* (*cf.* Figure 4.19). La description de cette composition (*OWL-S MyMeteo*), reposant sur OWL-S, est transmise au module de traduction (*OWL-S to SHOP2 translator*) qui se charge de convertir la description OWL-S en SHOP2 (*SHOP2 MyMeteo*). Cette nouvelle description est envoyée à la plate-forme. La plate-forme réalise ainsi le plan HTN de la composition *MyMeteo* (*MyMeteo-plan*) qui est transmis au module d'exécution du processus (*Execution Monitoring System*). Ce module se charge de l'invocation des services Web.

Les avantages de SHOP2 résident dans l'utilisation d'un HTN. D'une part, l'utilisation de ce concept issu du domaine de l'intelligence artificielle permet l'automatisation de l'exécution de la composition. D'autre part, la décomposition de tâche en sous-tâches afin d'appeler des services Web simples facilite la définition de la composition. Un des inconvénients de SHOP2 est que cette plate-forme ne permet pas la compensation de service si un service échoue. De plus, le fait que le module *Execution Monitoring System* conserve les résultats des services en mémoire durant l'exécution de la composition ne garantit pas la validité des données à un autre instant de cette même composition.

4.3.4 IRS-III

IRS-III (*Internet Reasoning Service*, version III) [Cabral *et al.*, 2006] est un cadre de travail pour l'implémentation d'applications à base de services Web sémantiques. Afin d'implémenter ce type d'application, IRS-III propose quatre étapes : l'annotation sémantique des services Web, la recherche et la sélection des services, et leur composition. Les deux premières étapes ont été étudiées dans la section 3.3.4 du chapitre 3. Dans cette section, nous nous consacrons à l'étude de la mise en œuvre de la composition.

L'architecture de IRS-III est composée de cinq composants mettant en œuvre la composition de services Web sémantiques (*cf.* Figure 4.20) : la bibliothèque de services Web sémantiques (*SWS Library*), l'interpréteur de chorégraphie (*Choreography Interpreter*), l'interpréteur d'orchestration

(*Orchestration Interpreter*), le gestionnaire de médiation (*Mediation Handler*) et le module d'invocation (*Invoker*).

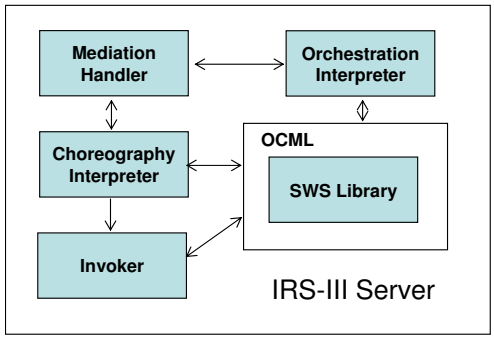


Figure 4.20 Architecture de IRS-III [Cabral *et al.*, 2006].

La bibliothèque de services Web sémantiques (*SWS Library*). Dans cette bibliothèque sont stockées les descriptions sémantiques formalisées en OCML (*Operational Conceptual Modelling Language*, langage de représentation sémantique issu des travaux de l'institut KMI – *Knowledge Media Institute*[60]) [Motta, 1999]. La bibliothèque est structurée en modèles de connaissances des *objectifs*, des *services Web* et des *médiateurs*. Ces modèles de connaissances sont utilisés afin de mettre en œuvre la résolution de problèmes, pour répondre à la requête de l'utilisateur (*cf.* section 3.3.4 chapitre 3).

L'interpréteur de chorégraphie (*Choreography Interpreter*). Dans IRS-III, la chorégraphie décrit les interactions entre la plate-forme et un service Web élémentaire et entre les services Web élémentaires [Domingue *et al.*, 2006]. La chorégraphie est représentée, d'une part, par un ensemble de règles de chaînage avant (*forward-chaining rule*), et, d'autre part, par une déclaration d'accès au service exprimée en OCML. Une règle exécute des interactions entre la plate-forme et le service Web si les conditions associées sont satisfaites. Ces interactions sont basées sur les primitives de communication suivantes : début de chorégraphie (*init-choreography*), envoi de message (*send-message*), réception de message (*receive-message*), réception d'erreur (*received-error*), et fin de chorégraphie (*end-choreography*).

L'interpréteur d'orchestration (*Orchestration Interpreter*). Dans IRS-III, l'orchestration est représentée par un modèle de *workflows* exprimé en OCML. La particularité d'utiliser OCML pour représenter un *workflow* est que l'unité de base d'une composition est un *objectif*. Le modèle fournit des constructions de flots de contrôle et de flots de données au-dessus de l'ensemble des objectifs. Les primitives de flots de contrôle disponibles dans IRS-III sont les suivantes : la liste des objectifs à invoquer de manière séquentielle (*orch-sequence*), une condition (*orch-if*), la répétition (*orch-repeat*), le résultat de la dernière invocation de l'objectif déclaré (*orch-get-goal-value*) et le résultat de l'exécution de l'objectif courant (*orch-return*).

Le gestionnaire de médiation (*Mediation Handler*). Ce composant associe à chaque objectif (représenté par une ontologie) un service Web. Cette association est permise par des règles de correspondance (*mapping rule*). La médiation intervient aussi bien lors de l'exécution de la chorégraphie que lors de l'exécution de l'orchestration dans le but de sélectionner, d'invoquer et d'exécuter les services Web adéquats.

Le module d'invocation (*Invoker*). Ce composant est l'interface de communication entre l'IRS-III et l'application réalisée à la suite de la requête du client. Il reçoit les entrées envoyées par le client et retourne les résultats de l'invocation au client.

[60] http://kmi.open.ac.uk/projects/ocml/

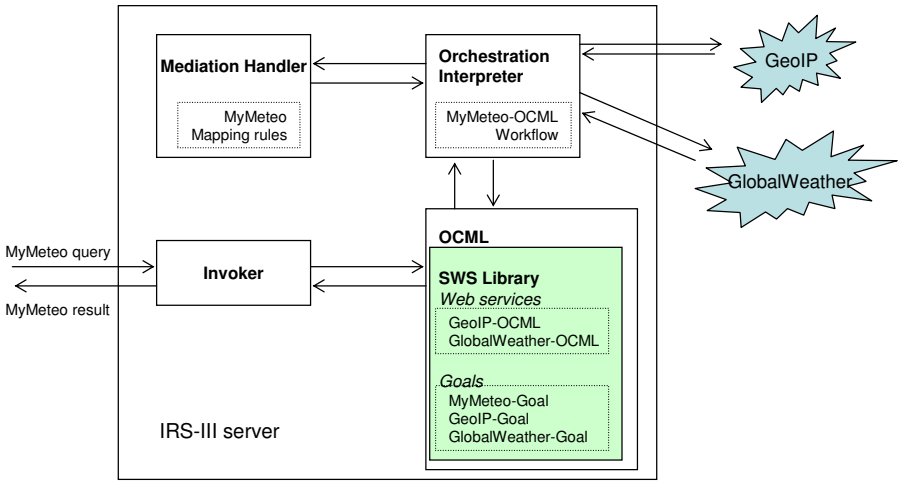


Figure 4.21 Illustration du fonctionnement d'IRS-III à l'aide de l'exemple de la composition *MyMeteo*.

Dans l'exemple de fonctionnement d'IRS-III (*cf.* Figure 4.21), nous faisons l'hypothèse que la composition *MyMeteo* est seulement décrite à l'aide de BPEL4WS et est ainsi de type orchestration. Par conséquent, l'interpréteur de chorégraphie (*Choreography Interpreter*) n'intervient pas dans notre exemple. Le module d'invocation (*Invoker*) reçoit la requête de la composition *MyMeteo* (*MyMeteo query*). La requête est transmise à l'interpréteur d'orchestration (*Orchestration Interpreter*) via la bibliothèque de services (*SWS Library*). La définition des flots de contrôle de *MyMeteo* est stockée dans ce module (*MyMeteo-OCML Workflow*). Via les règles de correspondance spécifiques à *MyMeteo* (*MyMeteo Mapping Rules*) et les bibliothèques de services et d'objectifs, l'interpréteur d'orchestration appelle les services adéquats. Le résultat de la composition (*MyMeteo result*) est transmis au module d'invocation qui l'envoie au client.

La particularité de cette plate-forme de composition de services Web est qu'elle autorise les deux types de composition (chorégraphie et orchestration). De plus, les auteurs utilisent des méthodes de résolution de problèmes pour réaliser la composition basée sur un objectif. Cependant, les services Web intervenant dans les compositions et la définition des *workflows* doivent être décrits en OCML. Or, au niveau de ce langage il n'existe pas de moyens qui permettent d'envisager des constructions évolutives de composition.

4.4 Conclusion

Dans notre travail, nous désignons la composition de services Web (quel que soit son type) comme un ensemble d'interactions entre services Web répondant à un problème complexe, formalisé par un client (logiciel ou humain) à l'aide d'une requête. Ces interactions (définies comme des échanges de données gérés par un flot de contrôle), ont pour but de réaliser un processus préalablement défini par les concepteurs. Une composition de services peut être de deux types : orchestration ou chorégraphie.

De nombreuses organisations (industrielles, telles qu'IBM et Microsoft, de recherche, telles que le W3C) travaillent afin de standardiser leur langage de composition de services Web. Dans ce chapitre, nous avons étudié trois langages de composition : BPEL4WS, WS-CDL et OWL-S.

BPEL4WS permet de définir une composition de services Web de type orchestration à l'aide de processus métier.

WS-CDL est langage de composition de type chorégraphie. Les interactions entre les services reposent sur les fondements du π-calcul [Milner, 1999].

OWL-S décrit les interactions entre les services Web sémantiques dans une composition de type chorégraphie à l'aide d'ontologies et de logiques de description.

Cependant, à ce jour, aucun de ces langages n'est considéré par la communauté des services Web comme un standard largement accepté pour la description de composition de services Web.

Un langage de composition de services Web permet uniquement aux concepteurs de systèmes à base de services de définir le processus de composition (l'enchaînement de l'appel de service dans le cas d'une orchestration et les échanges de messages entre deux services lors d'une chorégraphie). Afin de permettre la sélection des services, l'invocation des services, le contrôle de l'exécution, la compensation de services, les concepteurs de systèmes doivent utiliser des plates-formes de composition de services Web. Dans ce chapitre, quatre ont été étudiées (SELF-SERV, METEOR-S, SHOP2 et IRS-III).

SELF-SERV est une plate-forme de composition de services Web de type orchestration. La définition de la composition de services Web est décrite par l'intermédiaire de diagrammes d'états-transitions où chaque état peut être exécuté par une communauté de services Web. Cette idée permet de mettre en œuvre la compensation de services Web.

METEOR-S est une plate-forme de composition de type orchestration. La définition de la composition repose sur le langage BPEL4WS. Sa particularité est que les services Web invoqués, sont décrits à l'aide de OWL-S et sélectionnés dans un registre UDDI amélioré afin de gérer des services Web sémantiques.

SHOP2 est une plate-forme de composition de services Web sémantiques de type orchestration. La description de services Web sémantiques intervenant dans la composition doit être formalisée en OWL-S afin d'être interprétée par la plate-forme. La particularité de SHOP2 est que la définition de la composition est décrite en tant que décomposition de tâches à l'aide de l'HTN ce qui permet une automatisation de l'exécution de la composition.

IRS-III est un cadre de travail pour l'implémentation d'application à base de services Web sémantiques. Le processus d'implémentation est terminé par une composition de services Web. IRS-III gère aussi bien la composition de type chorégraphie (définie à l'aide de règles de chaînage avant et de déclaration d'accès au service) que la composition de type orchestration (définie à l'aide de *workflow*). La sélection des services à appeler est réalisée à l'aide de règles de correspondance entre les ontologies représentant la composition et celles représentant les services Web.

Lors de l'exécution de la composition, la plate-forme de composition doit, à chaque étape, sélectionner le service correspondant au mieux à la requête du client. Cependant, la plupart du temps, la requête seule ne permet pas de sélectionner le service Web qui convient le mieux. La prise en compte du contexte d'utilisation est nécessaire en vue d'améliorer les résultats de la composition de services Web.

5 ADAPTATION AU CONTEXTE DE SERVICES WEB

Ce chapitre est consacré à la mise en œuvre de l'adaptation au contexte dans les applications à base de services Web. Afin de répondre à cette problématique, il est, à notre avis, important de la prendre en compte dès le niveau conceptuel.

Dans le chapitre précédent, nous avons étudié des travaux portant sur la composition de services Web proposant des solutions (langages ou plates-formes) qui permettent de définir la composition afin de mettre en œuvre une application particulière. À chaque étape de la composition, le processus de sélection doit choisir le service Web qui convient le mieux à la requête du client. Afin que l'application soit adaptée au contexte d'utilisation (l'utilisateur, son dispositif d'accès, le lieu et le moment de l'utilisation de l'application), il est impératif, lors du processus de sélection, de prendre en compte ce contexte.

D'après [Benslimane *et al.*, 2007], une nouvelle génération de services Web, qui met l'utilisation du contexte au centre de leurs préoccupations, émerge. Ceci s'inscrit dans la nature dynamique de l'Internet en général et des services Web en particulier. Tout d'abord, les services Web élémentaires ont besoin d'une capacité à adapter leurs opérations afin de convenir à la situation dans laquelle ils sont appelés (*i.e.* le contexte). Ensuite, les services Web composants doivent pouvoir participer ou non à une composition de services Web selon le contexte. Or, les spécifications et langages sous-jacents aux services Web ne prennent pas en compte le contexte. D'après [Zhu *et al.*, 2005] et [Cuddy *et al.*, 2005], UDDI n'utilise pas les informations contextuelles dans le processus de recherche et ainsi ne renvoie pas les services les plus appropriés et pertinents pour les clients. De plus, les travaux de [Chen *et al.*, 2004] et [Wang *et al.*, 2004] montrent que le langage de description OWL-S n'inclut pas, dans sa forme originelle, de description sémantique du contexte.

De nombreux travaux proposent d'intégrer l'adaptation au domaine des services Web. Ces derniers présentent de multiples possibilités d'adaptation (telles que la personnalisation ou la recommandation de services) et interviennent dans différentes étapes du cycle de vie d'une application à base de services (telles que la sélection ou la planification). [Claro *et al.,* 2005] sélectionnent les services Web afin de les composer de manière optimale selon leur qualité de service – QoS (coût, temps d'exécution, disponibilité, et réputation). [Baresi *et al.,* 2003] sélectionnent les services Web (décrits selon leur qualité de service) répondant au mieux à la requête d'un client (profil de l'utilisateur et

localisation) selon des règles de correspondance. [Sam *et al.,* 2007] utilisent la logique d'attributs afin de décrire les services Web personnalisés et les requêtes des clients en vue de sélectionner les services répondant au mieux aux attentes des clients. [Taher *et al.,* 2008] utilisent le traitement d'événements complexes afin d'adapter les interfaces incompatibles entre les services communicants au sein d'une composition. [Soukkarieh *et al.*, 2008] proposent d'étendre l'architecture du système hypermédia adaptatif AHA ! [De Bra *et al.*, 2008] pour la recherche de services Web afin de concevoir des systèmes d'information basés sur le Web sensibles au contexte. Dans la suite de ce chapitre nous rapportons plus en détail les travaux qui portent sur l'adaptation au contexte d'utilisation que nous considérons significatifs dans le cadre de la problématique retenue dans cette thèse.

Dans ce chapitre, nous définissons tout d'abord l'adaptation au contexte d'utilisation. Ensuite, nous présentons les différents travaux qui mettent en relation l'adaptation au contexte d'utilisation avec, d'une part, les services Web élémentaires, et, d'autre part, la composition de services Web.

5.1 Adaptation au contexte

Cette section a pour but de présenter de manière plus détaillée la notion d'adaptation au contexte. Nous étudions tout d'abord les définitions des différentes familles de notion de contexte qui existent dans la littérature. Ceci définit le cadre général dans lequel nous analysons l'adaptation dans le domaine des services Web. Ensuite, nous présentons les objectifs de l'adaptation en décrivant trois domaines de l'informatique (l'informatique ubiquitaire, l'informatique pervasive, et l'informatique sensible au contexte) dont l'adaptation au contexte est l'une des principales préoccupations.

5.1.1 Définitions du contexte

De nombreux travaux issus du domaine de l'informatique sensible au contexte (dont les objectifs sont décrits au paragraphe suivant) tentent de donner une définition du contexte afin d'établir une base pour le processus d'adaptation. Selon [Tigli *et al.*, 2006], il existe quatre familles de contexte. Chacune de ces familles apporte leur propre définition du contexte. Les familles de contexte que nous mettons en évidence sont les suivantes : la notion de contexte généralisé, la notion de contexte géolocalisé, la notion de contexte environnemental et la notion de contexte unifié.

La notion de contexte généralisé. Certains auteurs ont tenté d'établir une définition générale du contexte afin de satisfaire la majorité des travaux issus du domaine de l'informatique sensible au contexte. Par exemple, [Gong, 2005] définit le contexte comme étant ce qui entoure et donne du sens à quelque chose d'autre. Cette définition sous-entend que le contexte est non seulement un monde d'objets et de relations, mais aussi qu'il est composé d'un objet de référence qui ancre le contexte. [Winograd, 2001] détermine le contexte comme un ensemble d'informations. Cet ensemble doit être structuré et partagé afin d'en faciliter l'exploitation, l'évolution et l'interprétation. Dans cette définition, l'ensemble des informations et son interprétation dépendent de la finalité des travaux utilisant le contexte.

La notion de contexte géolocalisé. Dans [Schilit *et al.*, 1994], les auteurs lient principalement le contexte à la géolocalisation de l'utilisateur. De ceci découlent trois types d'informations du contexte : (1) la localisation de l'utilisateur ; (2) les personnes se trouvant à proximité de l'utilisateur ; (3) les dispositifs et les ressources que l'utilisateur a à sa disposition.

La notion de contexte environnemental. Un grand nombre de travaux définissent le contexte comme étant l'environnement de l'utilisateur. Cet environnement permet de relier l'utilisateur à l'application. Dans [Brown *et al.*, 1997], les auteurs ajoutent à leur première définition du contexte (le contexte est l'ensemble des éléments de l'environnement de l'utilisateur [Brown,

1996]) des entités telles que l'heure, la saison, la température, l'identité et la localisation de l'utilisateur. Dans [Chen *et al.*, 2000], les auteurs définissent le contexte comme étant un ensemble d'états et de configurations observés de l'environnement. Ce contexte détermine, soit le comportement d'une application, soit les événements issus de l'application et qui intéressent l'utilisateur. Pour [Dey, 2001], le contexte est construit à partir de tous les éléments qui peuvent être utilisés pour caractériser la situation d'une entité. Une entité correspond ici à toute personne, tout endroit, ou tout objet (en incluant les utilisateurs et les applications eux-mêmes), considéré comme pertinent pour l'interaction entre l'utilisateur et l'application.

La notion de contexte unifié. Le contexte unifié est introduit par [Jang *et al.*, 2003]. Les auteurs spécifient le contexte comme étant une situation donnée par les réponses à cinq questions : Qui (*Who*) ?; Quoi (*What*) ?; Où (*Where*) ?; Quand (*When*) ?; Comment (*How*) ?; Pourquoi (*Why*) ? La réponse à la question « Qui ? » indique le sujet de l'action. Ce sujet peut être un utilisateur humain ou logiciel. La réponse à la question « Quoi ? » représente l'objet qui soutient l'action, tel que l'application utilisée. Les réponses aux questions « Où ? » et « Quand ? » donnent les informations spatio-temporelles à propos de l'action. La réponse à la question « Pourquoi ? » décrit les objectifs du sujet. La réponse à la question « Comment ? » explicite la manière dont l'action va être réalisée.

Les quatre familles de contexte décrites ci-dessus caractérisent un large panel de propositions de modélisation de contexte existantes. Tout d'abord, la notion de *contexte généralisé* permet d'englober la plupart des travaux. En effet, aujourd'hui des domaines d'une grande diversité (tels que l'intelligence artificielle, les sciences cognitives, la philosophie, etc.) travaillent sur la notion de contexte [Context, 1997]. Cependant, d'après [Power *et al.*, 2004], les auteurs doivent tenir compte du fait que dans la gestion du contexte qu'il n'y a pas qu'un modèle générique pour la représentation de l'information du contexte. Les définitions issues de la notion de *contexte géolocalisé* portent plus particulièrement sur l'environnement physique de l'utilisateur. Néanmoins, l'entité utilisateur ne permet pas à elle seule de définir, dans le domaine de l'informatique, la notion de contexte. La notion de *contexte environnemental* est celle qui est la plus largement adoptée dans le domaine de l'informatique sensible au contexte. Ces définitions ne nous satisfont pas entièrement puisqu'elles se limitent à une utilisation de l'application par un utilisateur humain. Or, l'utilisation d'un système peut être faite, soit par un utilisateur humain, soit par un autre système. Il faut alors prendre en compte ces deux types d'acteur. Enfin, la notion du *contexte unifié* englobe dans la définition du contexte le sujet de l'action, l'action elle-même, la localisation et le temps de l'action, l'objectif de l'action et sa réalisation.

Les notions et définitions du contexte étudiées ici sont indépendantes d'une formalisation spécifique. Cependant, lors de la mise en œuvre de l'adaptation au contexte, ces définitions du contexte peuvent être modifiées et améliorées selon le domaine et doivent être formalisées. Par exemple, [Lemlouma, 2004] propose d'adapter les documents multimédia en représentant le contexte à l'aide du langage CC/PP [Klyne *et al.*, 2004a] ; [Bucur *et al.*, 2005] utilisent les systèmes multi-agents afin d'adapter les applications mobiles au contexte d'utilisation, et représentent le contexte à l'aide d'ontologies et plus spécifiquement du langage OWL [McGuiness *et al*., 2004] ; [Kirsch Pinheiro, 2006] propose un formalisme du contexte à l'aide de la représentation de connaissances par objets afin d'adapter l'information de conscience de groupe dans les systèmes d'information coopératifs.

5.1.2 Objectifs de l'adaptation au contexte

La notion d'adaptation au contexte est au cœur des applications issues de trois domaines de l'informatique qui jouissent à présent d'une attention certaine de la part de la communauté

informatique : l'informatique ubiquitaire (ou ambiante), l'informatique pervasive et l'informatique sensible au contexte (*context-awareness computing*).

L'informatique ubiquitaire. [Weiser, 1991] est le premier à définir le terme d'informatique ubiquitaire. Cet auteur compare, à son époque, le passé de l'informatique – une machine (*mainframe*) pour plusieurs utilisateurs, le présent de l'informatique – un ordinateur pour un utilisateur (avec le PC – *Personal Computer*) et le futur de l'informatique – une multiplicité d'ordinateurs simplifiés et distribués dans l'environnement de l'utilisateur. Ce dernier aspect caractérise les principaux traits de l'informatique ubiquitaire. Le fait qu'il n'y ait pas que le domaine de l'informatique qui s'intéresse à l'informatique ubiquitaire (par exemple, le domaine de la Science de l'Information et de la Communication (SIC) [Barth, 2007], ou encore les institutions gouvernementales [MEIFT, 2006]) démontre l'importance que prend ce phénomène dans notre société.

> Dans le domaine de l'informatique ubiquitaire, l'objectif de l'adaptation au contexte est de **fournir aux utilisateurs des applications adaptées à la diversité des dispositifs qu'ils ont à leur disposition**.

L'informatique pervasive. D'après [Saha *et al.,* 2003], l'informatique pervasive tend à simplifier la vie des usagers par l'intermédiaire d'environnements digitaux qui capturent, qui s'adaptent, et qui répondent aux besoins de l'utilisateur. L'auteur compare l'informatique classique et l'informatique pervasive. Dans [Saha *et al.,* 2003] l'informatique classique est considérée comme l'exécution d'un programme dans un environnement virtuel pour exploiter les capacités d'un dispositif, tandis que les applications issues de l'informatique pervasive mettent en œuvre des solutions qui jouissent de l'ensemble des ressources disponibles de façon transparente pour l'utilisateur.

> Dans le domaine de l'informatique pervasive, l'objectif de l'adaptation au contexte est **d'utiliser le plus de ressources possibles localisées autour de l'utilisateur afin de lui fournir une application adaptée** sans qu'il intervienne dans le processus.

L'informatique sensible au contexte. L'informatique sensible au contexte est, d'après [Chen *et al.,* 2000], un paradigme de l'informatique mobile dans lequel les applications peuvent découvrir et utiliser l'information contextuelle. [Dey *et al.,* 2001] proposent trois étapes sur lesquelles doivent reposer les applications sensibles au contexte : (1) la capture des données contextuelles ; (2) la construction de l'information contextuelle à partir de l'interprétation des données capturées dans la première étape ; (3) la transmission de l'information interprétée à l'application et des différentes manières d'adapter l'application aux changements du contexte.

> Dans le domaine de l'informatique sensible au contexte, l'objectif de l'adaptation au contexte est de **fournir aux utilisateurs des applications adaptées qui prennent en compte le changement du contexte d'utilisation** inhérent au déplacement de l'utilisateur.

Cette étude a mis en évidence les besoins d'une définition assez large du contexte qui diffère selon le domaine dans lequel s'ancre l'adaptation. Cette définition doit être indépendante de tout formalisme, qui cependant doit exister afin que la définition du contexte soit exploitable et que les travaux apportent une forme d'adaptation à ce contexte. Dans le cadre de la mise en œuvre de l'adaptation dans le domaine des services Web (études des deux prochaines sections), ces travaux doivent, par conséquent, proposer une définition du contexte cohérente par rapport à leurs objectifs et une formalisation de la définition du contexte.

5.2 Approches d'adaptation au contexte pour les services Web élémentaires

Certains travaux rapprochent la technologie des services Web et l'adaptation au contexte. Cette section est consacrée aux travaux portant sur les services Web élémentaires. Différentes techniques sont utilisées afin d'intégrer le concept d'adaptation à la technologie de services Web (telles que l'adaptation du résultat du service, de la sélection ou de la composition). Nous présentons ici des travaux mettant en œuvre l'adaptation lors de la sélection de services Web [Balke *et al.,* 2003b], de l'exécution [Keidl *et al.,* 2004], de la présentation des résultats des services [Pashtan *et al.*, 2004], ou au niveau du langage de description sémantique OWL-S [Suraci *et al.*, 2007].

5.2.1 Sélection semi-automatique de services Web pour l'adaptation à l'utilisateur

Les travaux de [Balke *et al.,* 2003b] ont été décrits précédemment dans le cadre de l'étude sur les solutions de publication, de recherche et de sélection appropriées aux services Web sémantiques (*cf.* section 3.3). La particularité des travaux de ces auteurs est qu'ils intègrent l'adaptation au contexte (plus particulièrement à l'utilisateur) dans le processus de sélection de services Web.

Le processus de sélection de services Web se base sur la description de ces derniers. Dans ce travail, la description des services Web est une description sémantique exprimée à l'aide du langage DAML-S [Ankolekar *et al.*, 2002] (ancienne version de OWL-S [Martin *et al.*, 2004]). La description du service contient la description des interactions entre les acteurs (interaction avec un acteur logiciel – B2B ou humain – B2C), la description de la tâche (simple ou complexe) et son applicabilité (pré et post-conditions). Une tâche est dite complexe si elle a besoin d'informations supplémentaires (telles que le profil ou les préférences de l'utilisateur) afin d'être exécutée.

De manière simplifiée, le processus de sélection de services Web dans [Balke *et al.,* 2003b] repose sur trois étapes : l'expression de la requête de l'utilisateur, la recherche de services et la validation du résultat de la recherche par l'utilisateur.

L'expression de la requête. La requête de l'utilisateur est exprimée en SQL (*cf.* Figure 5.1). Afin de faciliter le processus de sélection [Balke *et al.,* 2003b] intègrent à la requête le profil de l'utilisateur (description de l'utilisateur incluant de l'information toujours vraie, telle que son nom, sa date de naissance) et les patrons d'utilisation (représentant les préférences de l'utilisateur pour un cas particulier) [Balke *et al.,* 2003a]. Dans l'exemple de requête illustrée par la Figure 5.1, l'utilisateur souhaite trouver un service Web lui proposant un restaurant de type `Casual`. L'utilisateur a une préférence pour les restaurants cantonais (concept `Cantonese`), puis chinois (concept `Chinese`) puis asiatique (concept `Asian`).

```
SELECT grounding(service)
FROM Restaurant
WHERE categogy = 'Casual'
PREFERING (concept(service) = 'Cantonese'
   PRIOR TO concept(service) = 'Chinese'
   PRIOR TO concept(service) = 'Asian')
```

Figure 5.1 illustration d'une requête de type SQL exprimée dans [Balke *et al.,* 2003b].

La recherche de services Web. La recherche est établie par la mise en correspondance des ontologies représentant la requête, le profil et les patrons d'utilisation et celles représentant les services.

La validation de la recherche par l'utilisateur. À la suite de la mise en correspondance, l'utilisateur déclare s'il est satisfait du résultat issu de la recherche. Si l'utilisateur n'est pas satisfait, le processus de recherche est renouvelé.

Le contexte est ici défini par un ensemble d'informations décrivant l'utilisateur. La définition du contexte se situe, d'une part, dans la description du service (profil et préférences de l'utilisateur présents lorsque le service propose une tâche complexe), et, d'autre part, dans la requête de l'utilisateur (profil et patron d'utilisation).

Une caractéristique intéressante de ce travail est qu'il fait intervenir l'utilisateur dans le processus d'adaptation pour sélectionner le service Web le plus adapté à son contexte (ici des données restreintes à l'utilisateur telles que son profil ou ses préférences). Cependant, en termes de représentation du contexte, la seule description de l'utilisateur est limitée. D'autres facteurs importants dans le processus d'adaptation, tels que le dispositif de l'utilisateur, sa localisation et l'environnement, ne sont pas pris en compte dans cette définition du contexte.

5.2.2 Adaptation de l'exécution des services

L'objectif des travaux [Keidl *et al.,* 2004] est d'adapter l'exécution des services Web selon un contexte prédéfini intégré dans le standard SOAP. Le processus d'adaptation, par l'intermédiaire du gestionnaire de contexte (*Context Manager*), s'effectue selon quatre étapes (*cf.* Figure 5.2) :

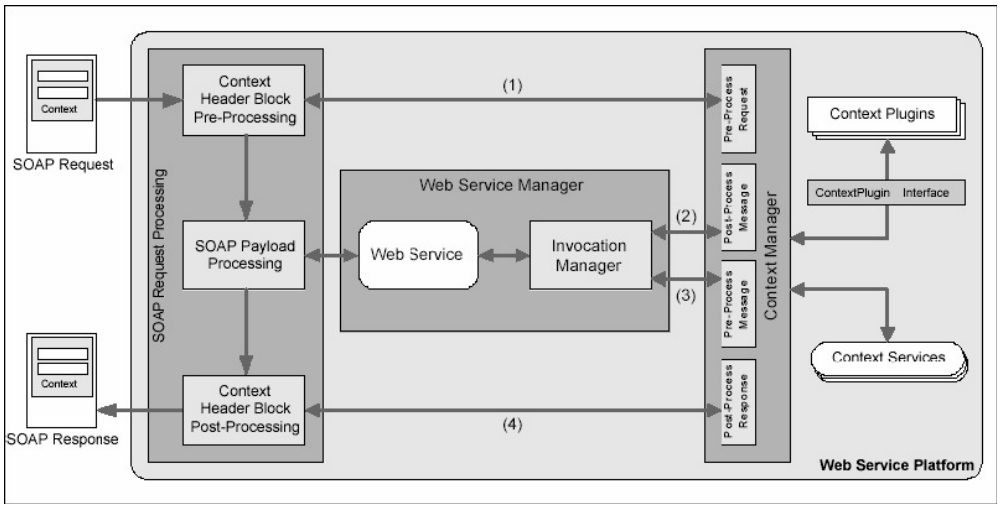


Figure 5.2 Composants pour l'adaptation des services Web par contexte, d'après [Keidl *et al.,* 2004].

Étape (1). La requête SOAP est pré-traitée par le gestionnaire de contexte (*Context Manager*, plus particulièrement le module *Pre-Process Request*) afin de récupérer le contexte inclus dans la requête. Ce processus permet de trouver à l'étape suivante un service Web adéquat.

Étape (2). Le gestionnaire de contexte invoque un service Web à l'aide du module *Invocation Manager*, dont le rôle est de sélectionner un service Web en adéquation avec le contexte défini dans la requête.

Étape (3). La réponse du service invoqué est transmise au gestionnaire de contexte afin de vérifier si elle est en adéquation avec le contexte.

Étape (4). Enfin, le résultat du service est envoyé au module de traitement SOAP (*SOAP Request Processing*) afin d'être traduit dans ce langage et être renvoyé au client.

Le contexte défini par [Keidl *et al.,* 2004] regroupe toutes les informations concernant le client qui sont nécessaires au processus d'adaptation des services Web. Ce contexte est caractérisé par cinq

éléments : la localisation, le dispositif, la présentation du résultat, l'utilisateur et les préférences de connexion.

La localisation. Ce premier élément permet de situer géographiquement l'utilisateur par l'intermédiaire d'une description donnée par l'utilisateur telle que « au bureau » ou « à la maison ».

Le dispositif. Cet élément comprend la description du terminal utilisé. [Keidl *et al.,* 2004] distinguent les caractéristiques matérielles (telles que le type de processeur, la mémoire libre, etc.), et les caractéristiques logicielles (telles que le système d'exploitation, le type de navigateur, etc.).

La présentation du résultat. Les auteurs caractérisent les propriétés de présentation du résultat du service Web retourné à l'utilisateur. Cet élément est représenté par l'intermédiaire de feuilles de style XSL (*eXtensible Stylesheet Language*) [Adler *et al.,* 2001] à appliquer aux résultats afin que ces derniers s'adaptent aux besoins ou restrictions imposées par l'utilisateur et son environnement.

L'utilisateur. [Keidl *et al.,* 2004] ajoutent à la définition du contexte des informations spécifiques à l'utilisateur (telles que son nom, son adresse électronique, etc.).

Les préférences de connexion. Le dernier élément concerne les préférences de connexion de l'utilisateur qui permettent de spécifier les propriétés de connexions aux services Web (telles que l'encodage du résultat – en format texte ou binaire –, la sensibilité du message – besoin d'encryptage ou non).

La mise en œuvre de l'adaptation dans les travaux de [Keidl *et al.,* 2004] est rendue possible en introduisant la définition du contexte dans les messages SOAP échangés entre le client et le service Web lors de l'appel du service. Ce contexte est traité par un gestionnaire de contexte afin de sélectionner un service Web en adéquation avec la requête. Nous pensons que le fait d'utiliser les standards des services Web dans le processus d'adaptation est justifié car cela permet de laisser les architectures de services Web existantes inchangées. Le fait d'utiliser SOAP pour transmettre le contexte courant de l'utilisateur est intéressant. Cependant, il manque dans ce travail une formalisation du processus de sélection du service Web. Les auteurs n'indiquent pas si la description classique des services Web, faite par WSDL, a été améliorée afin d'intégrer la description du contexte pour lequel le service peut apporter une adaptation ou si une autre méthode *ad-hoc* a été développée.

5.2.3 Conversion du résultat d'un service Web mobile pour l'adaptation de la présentation du résultat

Les travaux de [Pashtan *et al*., 2004], issus du domaine de la mobilité, adaptent au contexte les contenus générés par les services Web. Les auteurs proposent des services Web sensibles au contexte mobile dans le cadre d'une application touristique. Le contenu et la présentation des résultats des services Web sont adaptés aux dispositifs mobiles et à l'utilisateur.

Le processus d'adaptation est ici constitué de deux étapes (*cf.* Figure 5.3) [Pashtan *et al.,* 2004] : la génération du contenu en un format intermédiaire et une conversion du rendu final vers un langage de présentation spécifique.

Étape 1 (*Step 1*). Le résultat du service Web appelé est stocké dans une base de données (*Content Database*). Ce résultat est converti dans un format dit intermédiaire selon le contexte. À la suite de ce processus de conversion, le résultat est alors adapté au contexte. Il pourra, par exemple, être lisible sur un dispositif mobile alors que sa forme initiale ne permettait pas cette lisibilité. Le contexte est stocké dans trois bases de données : une base de données pour les

données courantes l'utilisateur, une seconde pour les données concernant la localisation de l'utilisateur et son environnement, et une dernière pour les données à propos des caractéristiques des dispositifs et du navigateur.

Étape 2 (*Step 2*). Le parseur XML DOM est utilisé pour convertir le résultat de la première étape (en HTML ou en WML) pour que la présentation du contenu du service Web corresponde aux caractéristiques du dispositif et au navigateur utilisé. Cette seconde conversion repose sur des feuilles de style XSLT [Clark, 1999].

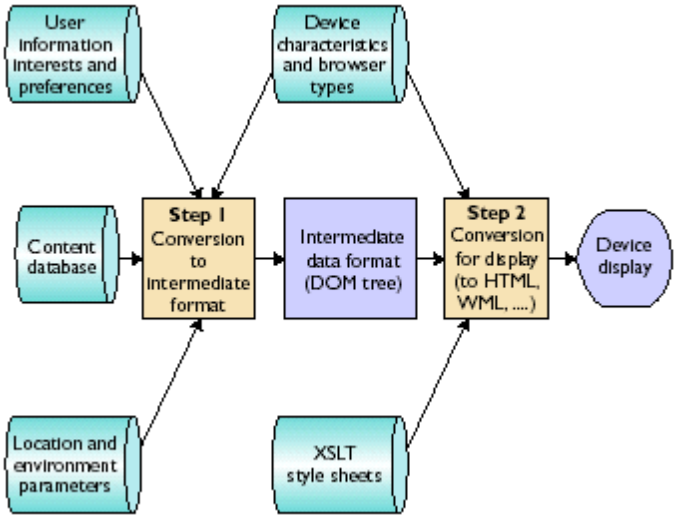


Figure 5.3 Conversion dynamique de contenu du résultat d'un service Web [Pashtan *et al.*, 2004].

La définition du contexte de [Pashtan *et al.*, 2004] comporte quatre catégories : les caractéristiques statiques de l'utilisateur, les caractéristiques dynamiques de l'utilisateur, les informations relatives à la connexion et à l'environnement.

Les caractéristiques statiques de l'utilisateur. Cette catégorie est composée de la description des intérêts et des préférences de l'utilisateur (par exemple, les préférences de l'utilisateur en termes de restaurants, hôtels ou tout autre besoin lié à une application touristique).

Les caractéristiques dynamiques de l'utilisateur. La localisation de l'utilisateur, ses tâches et ses activités constituent les caractéristiques dynamiques de l'utilisateur.

Les informations relatives à la connexion. Cette catégorie recouvre les capacités du dispositif, les caractéristiques du réseau, le type de navigateur utilisé et le coût de la communication.

Les informations relatives à l'environnement. Cette dernière catégorie décrit les données météorologiques, la lumière extérieure, le bruit environnant, la date et l'heure.

Du point de vue de la représentation du contexte, le fait de séparer les informations statiques et dynamiques est pertinent afin de mettre à jour les changements du contexte lors d'une même session de l'utilisateur. La mise en œuvre de l'adaptation dans les travaux de [Pashtan *et al.*, 2004], repose sur le fait que l'adaptation de la présentation est établie par des feuilles de style XSLT (*XSL Transformations*). Le contenu du service Web est converti afin d'être adapté au dispositif, selon le contexte. Nous regrettons dans ce travail la centralisation du processus d'adaptation. Si les étapes de conversion étaient effectuées, par exemple, par d'autres services Web, cette proposition d'adaptation aux dispositifs mobiles aurait pu être transposée à d'autres travaux.

5.2.4 Mise en œuvre de l'adaptation dans le cadre d'une recherche de services sensibles au contexte

[Suraci *et al.*, 2007] propose une solution pour découvrir des services Web sémantiques sensibles au contexte. Ce travail prend part au projet DAIDALOS II (*Designing Advanced network Interfaces for the Delivery and Administration of Location independent, Optimised personal Services*)[61], issu d'un programme de recherche européen. Pour ces auteurs, une recherche de services sensibles au contexte est définie comme la capacité d'utiliser l'information du contexte pour rechercher les services les plus pertinents pour l'utilisateur.

Le processus d'adaptation est ancré dans le mécanisme de recherche de services. Ce mécanisme se base sur l'architecture de référence des systèmes à base de services : le fournisseur publie ses services sur un serveur auquel l'utilisateur envoie sa requête de service. L'architecture de ce travail est illustrée par la Figure 5.4.

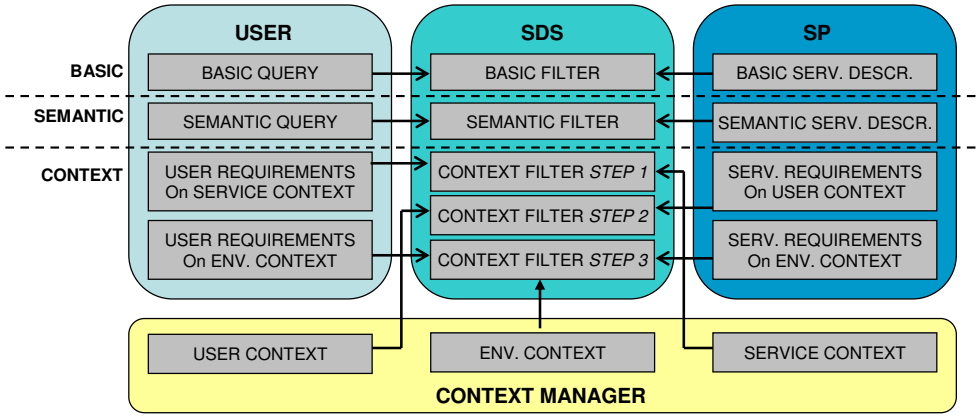


Figure 5.4 Architecture de recherche de services sensibles au contexte de l'utilisateur et illustration des trois mécanismes de filtrage, d'après [Suraci *et al.*, 2007].

Publication des services. Le fournisseur doit publier dans le gestionnaire de contexte (*Context Manager*, *cf.* Figure 5.4) le contexte auquel le service peut répondre et les conditions de son utilisation selon une description de l'utilisateur et de son environnement. La description du service Web publié peut se faire à deux niveaux. La première description est une **description basique** (*Basic Service Description*), exprimée dans un langage de bas niveau tel que XML pour les services non issus de la technologie Web, ou WSDL pour les services Web. La seconde description est une **description sémantique** (*Semantic Service Description*) exprimée en OWL-S. Le fournisseur doit publier dans le registre les deux descriptions du service (basique et sémantique), les conditions d'utilisation du service (*Service Requirements on User Context* et *Service Requirements on Environment Context*), et la référence du contexte du service. Ces informations sont stockées dans un module du serveur nommé *Service Provider – SP*.

Envoi d'une requête de services. Avant de pouvoir émettre une requête, l'utilisateur doit au préalable enregistrer son contexte (*User Context*) dans le gestionnaire de contexte (*Context Manager*). La requête de l'utilisateur est composée de deux parties : la requête basique (*Basic Query*) et la requête sémantique (*Semantic Query*). La **requête basique** est exprimée en langage de recherche de bas niveau (tel que – *Service Location Protocol* [Guttman *et al.*, 1999] ou UPnP[62] – *Universal Plug and Play*) et la **requête sémantique** est exprimée en langage de requête sémantique de haut niveau (tel que OWL-QL [Fikes *et al.*, 2003] ou RDQL [Seaborne,

[61] http://www.ist-daidalos.org/

[62] http://www.upnp.org/

2004]). L'utilisateur doit ensuite envoyer au serveur les requêtes et le pointeur sur son contexte. Ces informations sont stockées dans un module du serveur nommé *User*.

Une fois la requête de l'utilisateur reçue, le serveur de recherche de services (*SDS*) active le moteur de filtrage de services qui repose sur trois phases : le filtre de base, le filtre sémantique et le filtre de contexte (*cf.* Figure 5.4).

Le filtre de base (*Basic Filter*). Cette première étape utilise la requête basique de l'utilisateur et la description basique des services. Chaque description est comparée à la requête par l'intermédiaire du protocole SLP ou UPnP. Ce premier filtrage permet de sélectionner la catégorie du service à sélectionner et ainsi diminue le nombre de services qui vont être soumis à la seconde étape du filtrage.

Le filtre sémantique (*Semantic Filter*). Cette seconde étape compare la requête sémantique de l'utilisateur et la description sémantique des services sélectionnés lors de la première étape. Le résultat de ce filtre est une liste de services qui répondent exactement aux caractéristiques désirés par l'utilisateur, sans les informations du contexte.

Le filtre de contexte (*Context Filter*). Cette dernière étape est décomposée en trois phases. (1) Le serveur de recherche de service (*SDS*) compare le contexte du service et les attentes de l'utilisateur en termes de contexte de service. (2) Le SDS compare le contexte de l'utilisateur (défini plus bas) avec les conditions d'utilisation du service. (3) Le SDS compare le contexte de l'environnement (défini plus bas) avec les conditions d'utilisation du service et les attentes de l'utilisateur.

Le contexte est composé de deux catégories d'information relatives à l'**utilisateur** et à son **environnement**. La définition du contexte rassemble toutes les informations pour lesquelles l'utilisateur souhaite que le service puisse lui répondre de manière adaptée. Par exemple, en termes d'information sur l'**utilisateur**, le contexte peut contenir la position géographique de l'utilisateur, les informations concernant son dispositif, comme la charge de la batterie, ou la taille de l'écran. Concernant les informations relatives à l'**environnement**, la définition du contexte peut contenir la température, les accès de connexion, le taux de pollution, la détection de présence d'autres utilisateurs. En résumé, la définition du contexte dans ce travail rassemble tous les éléments qui peuvent être détectés de manière automatique par un capteur physique.

La particularité de ce travail en termes de représentation du contexte réside en deux points. Tout d'abord la description du contexte permet non seulement d'améliorer la requête de l'utilisateur mais aussi d'ajouter un nouveau niveau de sémantique à la description des services Web publiés. Ensuite, la représentation du contexte au sein de la description des services Web est permise par l'intermédiaire de l'extension de OWL-S. La description du contexte auquel peut répondre le service est formalisée en OWL. Les auteurs étendent le profil de service (*Service Profile*) de la description OWL-S (*cf.* section 2.2.2.1) par un attribut nommé `context` qui possède comme valeur l'URL de la description OWL du contexte.

Le fait d'intégrer le contexte auquel le service est adapté dans la définition du service est, de notre point de vue, une phase indispensable dans la mise en œuvre de l'adaptation dans le domaine des services Web. Un des inconvénients de ce travail est que le processus de filtrage n'est pas suffisamment flexible pour offrir une solution qui ne couvrirait qu'une partie du contexte. Ceci permettrait de rechercher plusieurs services recouvrant qu'une partie du contexte d'utilisation lorsqu'aucun service ne couvre la totalité de ce dernier.

Dans cette section, nous avons étudié quatre travaux portant sur l'adaptation au contexte de services Web élémentaires. Les **représentations du contexte** proposées par ces travaux intègre de

manière unanime l'**utilisateur**. Seul [Balke *et al.,* 2003b] n'inclut pas **l'environnement** dans la description du contexte. [Keidl *et al.,* 2004] et [Pashtan *et al*., 2004], issus du domaine de la mobilité, détaillent la représentation du contexte avec la description de la **localisation** de l'utilisateur, de son **dispositif**, de la **présentation du résultat** et des **préférences de connexion**. Dans ces travaux, la mise en œuvre de l'adaptation repose, soit sur la **sélection de services Web adaptés** ([Balke *et al.,* 2003b] et [Suraci *et al.*, 2007]), soit sur **l'adaptation du contenu** engendré par l'appel du service Web ([Keidl *et al.,* 2004] et [Pashtan *et al*., 2004]). La section qui suit étudie les approches d'adaptation pour la composition de services Web.

5.3 Approches d'adaptation au contexte pour la composition de services Web

Concernant la composition de services Web, de nombreux travaux intègrent, soit lors de la définition de la composition, soit lors de l'exécution de la composition de services, le concept d'adaptation au contexte. Nous étudions ici quatre techniques d'adaptation de la composition de services Web basées sur :

- Des **annotations du standard WSDL** [Mrissa, 2007] et de la **médiation**.
- La **description de services Web sensibles au contexte** dans le cadre d'une composition à l'aide de réseaux de tâches hiérarchiques [Vukovic *et al.,* 2004].
- L'**extension du langage de composition de services Web sémantiques OWL-S** [Ben Mokhtar *et al.,* 2005b] [Qiu *et al.,* 2007].

5.3.1 Annotation de WSDL et médiation pour la composition adaptée au contexte des services Web

Le travail de [Mrissa, 2007] résout les problèmes de compatibilité d'entrée-sortie entre les services au sein d'une composition de services Web en intégrant lors de l'exécution de la composition la notion de contexte. Afin d'atteindre son objectif, l'auteur se base sur la notion de médiateur. Dans une approche base de données, un médiateur a pour rôle de faire face à l'hétérogénéité entre des données provenant de sources différentes, afin de les traiter de manière uniforme [Wiederhold, 1992]. Pour [Mrissa, 2007] le rôle d'un médiateur dans le cadre d'une composition de services Web comprend l'adaptation des données échangées entre les services.

Ce travail s'intéresse uniquement au contexte d'une donnée. D'après [Mrissa, 2007], le contexte d'une donnée englobe tout élément interne ou externe relatif à la donnée, nécessaire à l'interprétation correcte de la donnée. Dans le cadre d'une composition de services Web, il existe deux types de contexte de la donnée : celui qui caractérise l'**émission de la donnée** (lié au service Web émetteur), et celui qui décrit l'**interprétation de la donnée** (lié au service Web destinataire). Pour représenter le contexte de la donnée, l'auteur définit un **objet sémantique** `S`, représentant la donnée, comme un quadruplet composé d'un concept `c`, d'une valeur `v`, d'un type `t` et d'un contexte `C` : `S=(c,v,t,C)`.

Le concept. Cet élément représente le concept auquel l'objet sémantique fait référence.

La valeur. Cet élément représente la valeur de l'objet sémantique.

Le type. Cet élément représente le type de la valeur décrite.

Le contexte. Cet élément représente le contexte de l'objet sémantique. Le contexte est composé de *modifieurs*. Un *modifieur* a la capacité de modifier la signification de l'objet sémantique (la

donnée) auquel il est associé. Par exemple, les *modifieurs* de l'objet sémantique *Prix* peuvent être : le pays d'origine, sa devise, si la TVA est incluse, et le facteur multiplicateur.

Le processus préalable à la composition de services Web et à l'adaptation de la composition est l'intégration du contexte des données au sein de la description des services Web. Cette description repose sur une annotation du langage de description WSDL (dans sa version 1.1 – *cf.* section 2.1.2.1). L'intégration du contexte dans WSDL repose sur l'ajout d'un attribut `contexte` au sein de l'élément représentant les paramètres (`part` dans la version 1.1 de WSDL – *cf.* section 2.1.2.1). Dans l'exemple présenté dans la Figure 5.5, le paramètre `inputPrice` de type `double` (ligne 3) a pour contexte celui de l'objet sémantique `Price` qui contient trois *modifieurs* (`France`, `VATIncluded` et `ScaleFactorOne`).

```
1  <wsdl:definitions> ...
2      <wsdl:message name="CarRentalTicket">
3          <wsdl:part name "inputPrice" type="xsd:double"
4                     ctxt:contexte="dom1:Price ctxt1:France ctxt1:VATIncluded ctxt1:ScaleFactorOne"/>
5      </wsdl:message>
6  </wsdl:definitions>
```

Figure 5.5 Annotation orientée contexte d'un paramètre d'une opération d'un service Web, d'après [Mrissa, 2007]

Le processus d'adaptation au contexte au sein de la composition de services Web repose sur l'intégration dans la composition d'un service Web dit médiateur. Ce service est inséré entre chaque couple de services Web de la composition. Le rôle du service médiateur est d'adapter les sorties du premier service aux entrées du second service. Le processus d'adaptation est réalisé en quatre étapes (*cf.* Figure 5.6) :

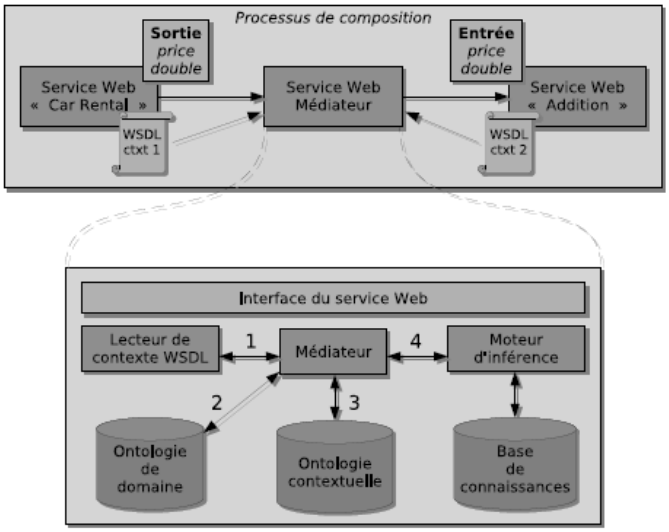


Figure 5.6 Vue détaillée du service Web médiateur, d'après [Mrissa, 2007].

Étape 1. Le service médiateur récupère les descriptions WSDL annotées des deux services Web.

Étape 2. Le service médiateur vérifie par l'intermédiaire des ontologies du domaine que les concepts utilisés par les deux services sont équivalents. Cette vérification d'équivalence consiste à déterminer si les contextes de données des paramètres de sortie du premier service correspondent aux contextes de données des paramètres d'entrée du second service. Si les concepts sont équivalents, l'exécution de la composition se fait de manière normale sinon le service médiateur effectue l'étape 3.

Étape 3. Pour chaque service, une structure arborescente du contexte de l'objet sémantique est construite à l'aide des ontologies contextuelles. Dans cet exemple, l'attribut `ctxt1:France` (*cf.*

Figure 5.5, ligne 4) devient une instance du concept `country` de l'ontologie contextuelle `ctxt1`, au sein de l'ontologie contextuelle non illustrée ici.

Étape 4. Afin de mettre en œuvre l'adaptation, le service médiateur effectue deux opérations à l'aide du moteur d'inférence. La première opération consiste en la déduction des valeurs des *modifieurs* en utilisant les règles logiques stockées dans la base de connaissances du moteur d'inférence. Par exemple, la description du contexte de l'objet sémantique `Price` contient le *modifieur* `France`. Par l'intermédiaire du moteur d'inférence, le service Web médiateur transmet au second service que la devise est l'`Euro`. La seconde opération consiste à convertir les données du premier service dans une représentation contextuelle requise par le second service. Si une valeur de *modifieur* manque, le moteur d'inférence recherche une règle définissant une valeur par défaut. Si aucune règle de ce type existe, la conversion est annulée, par conséquent la composition de services Web est levée.

Dans ce travail, la mise en œuvre de l'adaptation au contexte se positionne autour du contexte des données échangées entre les services Web. Cette notion de contexte ne prend pas en compte les définitions du contexte sous-jacentes aux environnements ubiquitaires, pervasifs et sensibles au contexte auquel nous nous intéressons. En effet, [Mrissa, 2007] ne fait pas référence aux attributs du contexte tels que la localisation de l'utilisateur et son environnement. La proposition de composition est décrite et validée dans le cadre d'une composition simple entre deux services. Cependant, si nous nous plaçons dans un contexte de composition complexe regroupant un ensemble plus important de services, le fait d'insérer entre chaque couple de services un service médiateur pourra, de notre point de vue, augmenter de manière importante les temps de réponse et les performances du système.

5.3.2 Composition adaptée contexte à l'aide de réseaux de tâches hiérarchiques

[Vukovic *et al.*, 2004] proposent une architecture pour réaliser des applications sensibles au contexte. Ces applications sont composées de séquences d'appels dynamiques aux services Web composants. La dynamicité découle des changements de contextes perceptibles durant la même session de l'application.

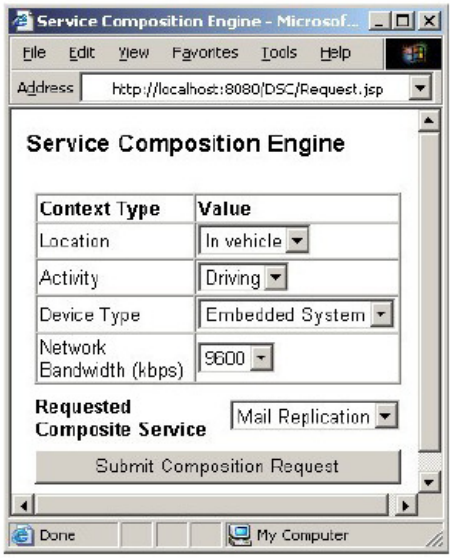


Figure 5.7 Illustration de la définition de la requête par l'utilisateur, d'après [Vukovic *et al.*, 2004].

La définition des séquences d'appels repose sur la planification de problème. Afin de définir une planification de problème, les systèmes se basent sur une **définition de problème** (*i.e.* la connaissance à propos de l'état du système et le but à atteindre) et une **définition du domaine** (*i.e.* les actions

disponibles et leurs conséquences). [Vukovic *et al.*, 2004] utilisent la plate-forme SHOP2 (*cf.* section 4.3.3) afin de mettre en œuvre la composition de services Web à l'aide de la planification de problème.

L'étape préliminaire au processus d'adaptation et de composition est l'**envoi de la requête de composition de services**. La requête de l'utilisateur est composée de données sur son contexte (défini plus bas) et sur son objectif. L'utilisateur transmet au système manuellement ces données par l'intermédiaire d'un formulaire. Dans l'exemple de définition de requête illustré par la Figure 5.7, l'utilisateur souhaite une composition de services Web répondant à un objectif (*reproduction d'e-mails*) adaptée à un contexte prédéterminé par sa localisation (*dans son véhicule*), son activité (*en train de conduire*), le type de dispositif (*système embarqué*) et la bande passante du réseau (*9600 kbps*).

Dans [Vukovic *et al.*, 2004], les processus de composition des services Web et d'adaptation au contexte comportent trois étapes : la définition du problème, la définition du domaine et la conversion en BPEL4WS (*cf.* Figure 5.8).

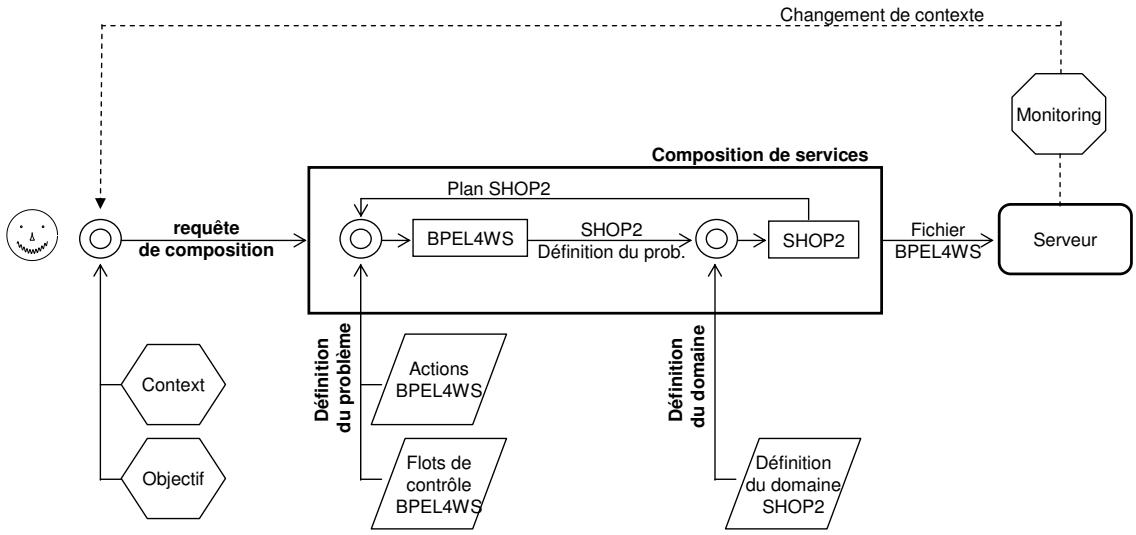


Figure 5.8 Architecture du système pour la composition dynamique de services sensibles au contexte, d'après [Vukovic *et al.,* 2004].

La définition du problème. Pour définir les problèmes en SHOP2, le système utilise des flots de contrôle et des actions décrits en BPEL4WS. Ces fichiers BPEL4WS sont décrits préalablement à l'envoi de la requête.

La définition de domaine. La définition de domaine (les services Web disponibles) est connue au préalable et pré-compilée en SHOP2 en y incluant les opérateurs entre les services. Dans la définition de domaine, les services Web sont sélectionnés selon le contexte de l'utilisateur. Les définitions du problème et de domaine sont compilées par JSHOP [Nau *et al.*, 2003] (une implémentation Java de SHOP – système de planification de problèmes [Nau *et al.*, 2001]) afin d'avoir en retour la définition des plans en SHOP2.

La conversion en BPEL4WS. Les plans pré-définis sont convertis en BPEL4WS de manière semi-automatique (chaque opérateur de SHOP2 correspondant à une définition d'activité BPEL4WS). Le fichier BPEL4WS décrivant la composition de services Web est transmis au serveur de l'application.

La dernière étape est l'**exécution des services**. Le fichier BPEL4WS, une fois sur le serveur, est déployé par BPWS4J (*Business Process Execution Language for Web Services Java*)[63]. L'utilisateur

[63] http://www.alphaworks.ibm.com/tech/bpws4j

déclenche ensuite l'exécution de la composition par l'intermédiaire d'une interface graphique qui lui est proposée par l'application Web.

La définition du contexte dans ce travail couvre trois entités : l'utilisateur, le réseau, et le dispositif.

L'utilisateur. Cette entité est composée des informations concernant l'activité de l'utilisateur (ce que l'utilisateur fait lors de l'envoi de la requête, tel que « en train de marcher », « en train de conduire », « en train de travailler ») et la localisation de l'utilisateur (tel que « dans la voiture », « au bureau »).

Le réseau. Cette entité comporte les informations telles que le type de réseau disponible et la bande passante.

Le dispositif. Cette entité décrit les informations concernant le dispositif de l'utilisateur telles que la taille de l'écran ou la profondeur de couleurs pouvant être prises en compte par l'écran.

L'originalité de ce travail est l'utilisation de la planification de problèmes pour définir la composition de services Web sensibles au contexte. Cependant, le formalisme et la définition exacte du contexte ne sont pas décrits en détail. De plus, les valeurs des éléments du contexte reste floues, comme avec par exemple les valeurs de l'élément *localisation de l'utilisation* qui peuvent être dans ce travail « dans la rue », « en voiture » ou « au bureau ». Le fait que ce soit l'utilisateur qui déclenche l'exécution des services Web est surprenant étant donné que le résultat de la requête est censé être sensible au contexte. Ce travail se dit réactif au changement du contexte. Or, s'il se produit un changement de contexte, une nouvelle requête doit être transmise au système par l'utilisateur et le processus de composition et d'adaptation doit être effectué à nouveau dans son intégralité.

5.3.3 Extension de OWL-S

Dans cette sous-section nous avons choisi de décrire deux travaux [Ben Mokhtar *et al.*, 2005b] [Qiu *et al.*, 2007] qui utilisent la même stratégie d'intégration de la définition du contexte dans les services Web : par extension du langage OWL-S. Cependant, ces deux travaux utilisent des méthodes différentes pour adapter la composition de services Web au contexte : soit une approche par automates d'états finis dans les systèmes pervasifs [Ben Mokhtar *et al.*, 2005b], soit une approche par planification [Qiu *et al.*, 2007].

5.3.3.1 Approche par automates d'états finis

[Ben Mokhtar *et al.*, 2005b] apportent une solution pour composer des services sensibles au contexte pour des systèmes pervasifs, selon la tâche de l'utilisateur.

La définition du contexte englobe dans ce travail la **description des capteurs de contexte**, des **services**, des **dispositifs** et des **utilisateurs**. Chacune de ces entités est décrite par une ontologie. L'étape préliminaire au processus d'adaptation est la représentation du service et de la tâche de l'utilisateur par l'intermédiaire de l'extension de OWL-S que proposent les auteurs.

Extension de OWL-S pour représenter le service. Cette extension permet de décrire les propriétés non fonctionnelles du service. L'extension de OWL-S touche l'ontologie *Service Profile* et l'ontologie *Service Model* (*cf.* section 2.2.2.1). L'extension de l'ontologie *Service Profile* intègre de nouveaux concepts décrivant les attributs du contexte qui caractérisent le service de manière isolée. L'extension de l'ontologie *Service Model* intègre les attributs de qualité de service en plus de la description des pré-conditions, et des effets du service proposées dans la version originelle de OWL-S.

Extension de OWL-S pour représenter la tâche de l'utilisateur. Cette extension permet de décrire la tâche de l'utilisateur. Elle concerne l'ontologie *Service Model* dans laquelle elle intègre les descriptions des conditions de qualité de service et les conditions du contexte requises par la tâche de l'utilisateur, représentées par des ontologies.

Le processus d'adaptation intégré à la composition de services Web se déroule en deux étapes : la sélection des services Web, et la définition de la composition de services.

Sélection des services Web. La sélection des services Web est réalisée par la mise en correspondance des extensions de l'ontologie *Service Model* de OWL-S (représentant la tâche de l'utilisateur et le service) et des conditions contextuelles de la tâche (extension du *Service Model*) avec les attributs du contexte du service (extension du *Service Profile*).

Définition de la composition de services Web. [Ben Mokhtar *et al.*, 2005b] adoptent l'approche par automates d'états finis pour définir la composition de services Web [Ben Mokhtar *et al.,* 2005a]. Chaque *Process Model* de OWL-S (que ce soit celui du service ou celui de la tâche de l'utilisateur) est représenté par un automate d'états finis. La composition est réalisée en trois étapes.

(1) L'ensemble des automates décrivant chaque service Web sélectionné est assemblé afin de former un automate global.

(2) Pour chaque état de l'automate représentant la tâche, le système recherche un état équivalent dans l'automate global.

(3) Pour mettre en œuvre l'adaptation au contexte, à chaque mise en correspondance, les pré-conditions et les effets des services ainsi que les conditions contextuelles de la tâche de l'utilisateur sont vérifiés.

L'originalité de ce travail réside dans l'utilisation d'automates d'états finis pour représenter la composition de services Web. Cependant, l'utilisation de OWL-S pour décrire le contexte de la tâche de l'utilisateur détourne l'objectif premier de OWL-S qui est de décrire des services Web sémantiques.

5.3.3.2 Approche par planification

[Qiu *et al.*, 2006], [Qiu *et al.*, 2007] proposent une méthode de planification sensible au contexte pour la composition des services Web. La particularité de ce travail est qu'il comprend non seulement une planification globale selon la requête de l'utilisateur, mais aussi une optimisation locale selon les changements du contexte. Cette proposition repose, d'une part sur une extension de OWL-S, nommée OWL-SC (*OWL-S Context*) permettant d'intégrer le contexte à la description des services Web, et, d'autre part, sur la planification basée sur le contexte pour la composition de services.

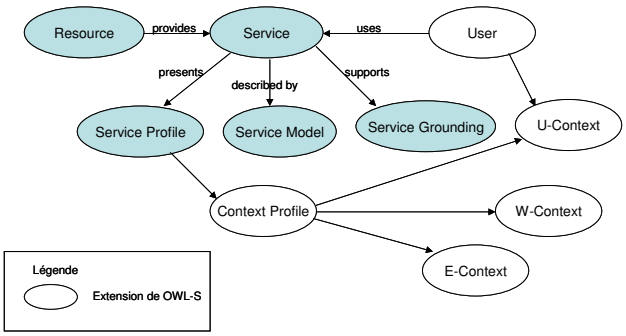


Figure 5.9 OWL-SC, extension de OWL-S : ajout des ontologies de contexte dans la description de services Web, d'après [Qiu *et al.,* 2006].

Dans [Qiu *et al.*, 2006], les auteurs définissent trois types de contexte : le contexte de l'utilisateur (*U-Context*), le contexte du service Web (*W-Context*) et le contexte de l'environnement (*E-Context*). Chacun de ces contextes est représenté par une ontologie et est intégré aux ontologies existantes de OWL-S (*cf.* Figure 5.9).

U-Context. Ce contexte est constitué d'informations concernant l'utilisateur, plus spécifiquement où se trouve l'utilisateur et ce qu'il fait. Les auteurs divisent cette information en deux contextes : le contexte statique de l'utilisateur (son profil, ses intérêts, ses préférences) et le contexte dynamique de l'utilisateur (sa localisation, son activité courante et la tâche qu'il est en train d'effectuer).

W-Context. Ce contexte regroupe les informations non fonctionnelles d'un service, telles que son prix, la durée de son exécution, son degré de confiance.

E-Context. Ce contexte comprend l'information à propos de ce qui entoure l'utilisateur, telle que le temps et la date.

Le processus d'adaptation au contexte intégré à la composition de services Web est basé sur un déroulement en trois étapes (la gestion du contexte, la planification et l'exécution) – *cf.* Figure 5.10.

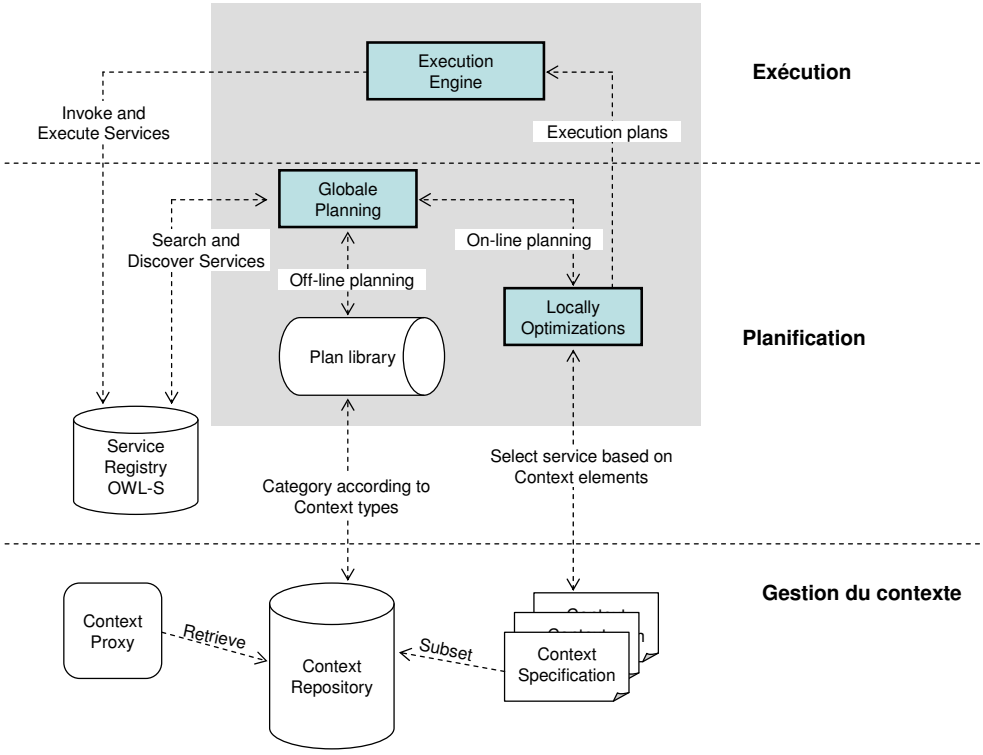


Figure 5.10 Architecture proposée par [Qiu *et al.,* 2006] pour composer des services Web adaptés au contexte d'utilisation, d'après [Qiu *et al.,* 2006].

La gestion du contexte. Trois composants permettent de gérer le contexte dans le travail de [Qiu *et al.,* 2006] : le *Context Proxy*, le *Context Repository* et le *Context Specification* Le *Context Proxy* recherche les informations du contexte et génère les conditions du contexte qui doivent être utilisées lors de la requête de la composition. Le *Context Repository* stocke les représentations des contextes (*U-Context*, *W-Context* et *E-Context*) formalisées dans leur extension de OWL-S nommée OWL-SC. Le *Context Specification* représente des règles pré-définies associées aux conditions de contexte pour le filtrage de services.

La planification. Selon la requête de l'utilisateur et les informations du contexte, le système définit une séquence d'actions qui, une fois exécutée, produit le résultat de la requête initiale. La

base de données de plans (*Plan Library*) stocke des schémas de plans, représentés par des diagrammes d'états UML, définis par le concepteur hors-ligne. Ces schémas peuvent être ensuite exécutés dans leur globalité par le composant *Global Planning* qui, pour chaque état, cherche un service Web dans le registre de services. Si, dans les schémas, il existe des plans multiples, le composant *Locally Optimisation* sélectionne le plan adéquat selon le contexte.

L'exécution. Une fois le plan choisi et les services Web sélectionnés le plan est exécuté par le moteur d'exécution (*Execution Engine*).

Ce travail propose une méthode de planification pour composer des services Web adaptés au contexte. L'extension de OWL-S qui permet de décrire le contexte pour lequel le service Web peut répondre est intéressante et permet de prendre en compte une large définition du contexte (utilisateur, les données non fonctionnelles du service et l'environnement de l'utilisateur). Le fait de définir la planification hors-ligne et d'en proposer des optimisations selon le contexte lors de l'envoi de la requête est, de notre point de vue, pertinent. Cependant, le fait que ce soit le concepteur qui définisse lui-même de manière statique les schémas de planification est assez restrictif et ne garantit pas une évolution dynamique du système lors de l'intégration de nouveaux services.

5.4 Conclusion

Ce chapitre nous a permis de mettre en relief deux points importants qui concernent l'intégration de l'adaptation au contexte à la technologie des services Web. Tout d'abord, nous avons décrit les différentes notions du contexte rencontrées dans la littérature. Ensuite, nous avons identifié un ensemble de types et techniques d'adaptation spécifiques à l'utilisation des services Web.

Trois domaines de l'informatique (informatique ubiquitaire, pervasive et sensible au contexte) en pleine évolution montrent de l'intérêt pour la prise en compte du contexte afin d'adapter les applications. De nombreux travaux ont adopté leur propre notion du contexte. Il existe quatre notions de contexte (contexte généralisé, géolocalisé, environnemental et unifié). Chaque travail portant sur l'adaptation au contexte doit, au préalable, donner une définition du contexte pour lequel il apporte une adaptation. Ces définitions s'inscrivent dans les notions de contexte citées ci-dessus. Chaque travail choisit ensuite librement sa propre formalisation du contexte.

À la suite de cette étude, nous remarquons qu'il est important, afin de mettre en œuvre l'adaptation dans le cadre de services Web, d'intégrer à la description des services la définition du contexte d'utilisation couvert par le service [Suraci *et al.*, 2007]. Cette intégration peut être réalisée par annotation sémantique du langage standard WSDL [Mrissa, 2007], ou par extension du langage OWL-S [Ben Mokhtar *et al.*, 2005b] [Qiu *et al.*, 2006].

L'étude des travaux présentés dans ce chapitre nous a permis de déterminer trois dimensions d'adaptation dans le domaine des services Web (l'adaptation du résultat du service, l'adaptation de la demande de services et l'adaptation de la composition).

L'adaptation du résultat du service. L'adaptation du résultat du service propose d'adapter le résultat retourné au client (dans le cadre d'un appel à un service Web élémentaire) [Pashtan *et al.*, 2004] ou au service Web suivant dans la composition (dans le cadre de la composition de services Web) [Mrissa, 2007]. La technique d'adaptation mise en jeu pour atteindre cette dimension d'adaptation est la **conversion du résultat**.

L'adaptation de la demande de services Web. Lors de l'appel d'un service Web élémentaire, certains travaux intègrent le contexte d'utilisation afin de trouver le service Web le mieux adapté à la demande du client [Balke *et al.,* 2003b] [Keidl *et al.,* 2004] [Suraci *et al.*, 2007]. Le système peut avoir à sa disposition un ensemble de services Web qui peut répondre de manière

fonctionnelle à la demande. La technique d'adaptation mise alors en jeu est celle de **sélection de services Web** qui consiste à sélectionner le service qui correspond au mieux au contexte.

L'adaptation de la composition. D'après les travaux étudiés dans ce chapitre, l'adaptation de la composition au contexte peut s'établir par l'intermédiaire de deux techniques d'adaptation. Tout d'abord, s'il existe plusieurs définitions de composition, le système peut **sélectionner la composition** répondant au mieux au contexte [Vukovic *et al*., 2004] [Qiu *et al.*, 2006]. Ensuite, une **sélection de services Web** selon le contexte peut être faite à chaque étape de la composition [Ben Mokhtar *et al.*, 2005b].

6 SYNTHÈSE DE L'ÉTAT DE L'ART

Ce chapitre propose une synthèse des études effectives dans l'état de l'art. Cette synthèse permet de mettre en évidence les éléments que nous devons apporter afin de faciliter la tâche des concepteurs lors de la mise en œuvre (conception, implémentation) d'applications adaptées à base de services Web.

Dans un premier temps, les études réalisées dans les chapitres 2 et 3, portant sur les travaux concernant la description, la publication, la recherche et la sélection des services Web, permettent de dresser un tableau comparatif des langages et outils proposant de représenter les services Web. Dans un second temps, nous confrontons les solutions de publication de services Web et leurs méthodes de recherche (synthèse du chapitre 3). Puis, nous établissons un bilan des plates-formes de composition de services Web selon les solutions apportées et les méthodes utilisées (synthèse du chapitre 4). Ensuite, les travaux abordant l'adaptation au contexte d'utilisation dans le domaine des services Web sont rapprochés (synthèse du chapitre 5). Enfin, nous décrivons les objectifs de notre proposition qui reposent sur les points mis en exergue dans cette synthèse des études faites dans l'état de l'art.

6.1 Représentation de services Web

La représentation des services Web est constituée d'un ensemble de catégories d'information qui caractérise un service Web lors de sa description et de sa publication par le fournisseur. Cette représentation est importante en vue de faciliter les étapes de recherche et de sélection par le futur client.

Afin d'utiliser et de réutiliser les services Web, le fournisseur doit représenter en premier lieu les **aspects fonctionnels** du service comme cela a été présenté dans le chapitre 2. Ces aspects ne suffisent cependant pas pour faciliter les tâches de recherche et de sélection de services Web. C'est pourquoi les travaux portant sur la publication, la recherche, et la sélection (*cf.* chapitre 3) enrichissent la représentation fonctionnelle des services Web de trois catégories d'information. Tout d'abord, les travaux décrivent le **fournisseur**. Ensuite, certains travaux ajoutent à la représentation du service la description de la **tâche** qu'il effectue. Cette description est composée du domaine et de la catégorie auxquels appartient le service, ainsi que son (ou ses) objectif(s). Enfin, des travaux ajoutent la description de l'**utilisation du service**, à l'aide de la description de la qualité de service et des pré et post conditions de ce dernier.

Cette représentation enrichie de services Web peut être réalisée par divers moyens. Elle peut être faite par le biais, soit de descriptions textuelles, soit de langages semi-structurés (tels que XML), soit de langages issus du Web sémantique (tels que OWL, RDF ou OWL-S). Ce dernier type de langage à l'avantage d'apporter par la suite des mécanismes puissants d'interprétations tels que les inférences.

Le Tableau 6.1 met en évidence ces catégories d'information et les travaux proposant des solutions de description, publication, recherche, et sélection de services Web qui utilisent ces catégories pour enrichir la représentation des services Web.

Légende :
'+' catégorie prise en compte
'++' catégorie prise en compte par des langages du Web sémantique
'-' catégorie non prise en compte

		Aspects fonctionnels	Fournisseur	Description de la tâche du service			Description de l'utilisation du service	
				Domaine du service	Catégorie du service	Objectif du service	Qualité de service	Pré-Post conditions
Langage de description	WSDL	+	-	-	-	-	-	-
	OWL-S	+	++	-	-	++	-	++
	SAWSDL	++	-	-	-	++	-	-
Registre accessible via le Web	UDDI	+	+	-	+	-	-	-
	XMethods	+	+	-	-	+	-	-
	RemoteMethods	+	+	+	+	+	+	-
	QWSDataset	+	-	-	-	-	-	-
Solution de publication, recherche et sélection	Découverte coopérative [Balke *et al.*, 2003b]	++	-	-	-	-	-	++
	ASSAM [Heß *et al.*, 2004]	++	-	++	++	-	-	-
	Approche basée sur une logique de description [Baader *et al.*, 2005]	++	-	-	-	-	-	++
	IRS-III [Cabral *et al.*, 2006]	++	-	-	-	++	-	-

Tableau 6.1 Catégories d'information relatives aux services Web proposées par les travaux portant sur la publication, recherche et sélection de services Web.

Les aspects fonctionnels. Pour décrire les aspects fonctionnels des services Web, le langage standard WSDL est incontournable. Par l'intermédiaire du langage semi-structuré XML, WSDL permet aux fournisseurs de décrire : les **types de données**, les **opérations**, les **messages**, le **protocole de communication** et les **ports d'accès** des services Web. SAWSDL permet d'ajouter un niveau sémantique, à l'aide d'annotations, aux aspects décrits par le standard (c'est-à-dire les aspects fonctionnels).

Les aspects liés au fournisseur. Les informations concernant le fournisseur (telles que son site Web, le nom du responsable de la maintenance des services Web) sont importantes lors de l'étape de sélection, mais aussi nécessaires si le client rencontre des problèmes avec le service invoqué. Ces aspects sont pris en compte par UDDI et les registres publics *XMethods* et *RemoteMethods*. OWL-S propose, à l'aide d'ontologies, de décrire le fournisseur du service Web afin de mettre en évidence, par exemple, des catégories de fournisseur.

Les aspects liés à la tâche du service Web. Afin de sélectionner un service, les seuls aspects fonctionnels ne suffisent pas. Les clients ont besoin de connaître la tâche du service par l'intermédiaire de descriptions :

- du **domaine d'application du service**, par exemple le domaine de la géographie (pris en compte par le site *RemoteMethods* et par le biais de langages du Web sémantique – tels que OWL-S, par ASSAM [Heß *et al.¸2004]*) ;
- de la **catégorie à laquelle appartient le service**, par exemple les services de localisation (prise en compte par UDDI, *RemoteMethods* et par le biais OWL-S par ASSAM) ;
- de l'**objectif du service**, *i.e.* son objectif, par exemple un service Web qui, à partir de coordonnées GPS, renvoie la ville où l'utilisateur se situe (pris en compte par *XMethods*, *RemoteMethods* et par le biais de langages du Web sémantique – tels que WSMO, par IRS-III [Cabral *et al.,* 2006]). Concernant le langage de description de services Web, OWL-S décrit l'objectif du service Web à l'aide d'ontologies, tandis que SAWSDL utilise des annotations sémantiques.

Les aspects liés à l'utilisation. Nous déterminons deux catégories d'information relatives aux aspects d'utilisation d'un service Web. Tout d'abord la catégorie d'information représentant la **qualité de service**. Dans *RemoteMethods*, cet aspect repose sur les commentaires de développeurs consécutifs à une utilisation des services Web. Dans *QWSDataset*, la qualité de service est représentée par un ensemble d'informations déterminées par [Ran, 2003] et [Kalepu *et al.,* 2003] (telles que le temps de réponse, la fiabilité) dont la valeur est donnée suite à plusieurs invocations automatiques de services. La seconde catégorie d'information qui compose les aspects liés à l'utilisation est constituée de **pré et post-conditions** à l'utilisation des services. Cette catégorie est prise en compte de manière sémantique tant par [Balke *et al.,* 2003b] à l'aide d'ontologies, que par [Baader *et al.,* 2005] à l'aide d'une logique de description. OWL-S décrit à l'aide d'ontologies les pré et post-conditions du service.

En résumé, selon l'ensemble des travaux que nous avons étudié en termes de description, publication, recherche et sélection de services Web, quatre aspects doivent être pris en compte au sein de la représentation de services Web en vue de faciliter l'implémentation d'applications à base de services Web : les **aspects fonctionnels**, les **aspects liés à la tâche du service**, les **aspects d'utilisation** et les **aspects liés au fournisseur**. Aucun travail étudié ne vise l'ensemble de ces aspects.

6.2 Solutions spécifiques à la publication et à la recherche

Dans le chapitre 3, nous avons étudié un ensemble de travaux proposant des solutions de publication et de recherche de services Web.

La **publication** est une étape importante dans le cycle de vie d'une application à base d'architecture orientée services. En effet, cette étape permet aux fournisseurs de rendre visibles leurs services Web afin qu'ils soient utilisés (et réutilisés) par le plus grand nombre de clients. Une fois ces services publiés dans les registres, ces derniers doivent être dotés de **méthode de recherche** afin que les clients recherchent et sélectionnent facilement les services dont ils ont besoin.

Le Tableau 6.2 dresse un bilan de ces solutions. Il met en évidence tout d'abord les travaux qui proposent, ou non, une possibilité aux fournisseurs de publier leurs services, puis, les méthodes de recherche que ces solutions offrent aux clients.

		Publication oui/non	Méthode de recherche
Registre accessible via le Web	UDDI	oui	- Recherche par mot-clé
	XMethods	oui	- Aucune
	RemoteMethods	oui	- Navigation par catégories
	QWSDataset	non	- Recherche par mots-clés
Solution de publication, recherche et sélection	Découverte coopérative [Balke *et al.*, 2003b]	non	- Alignement d'ontologies
	ASSAM [Heß *et al.*, 2004]	oui	- Construction de *clusters*
	Approche basée sur une logique de description [Baader *et al.*, 2005]	oui	- Inférence
	IRS-III [Cabral *et al.*, 2006]	oui	- Résolution de problème

Tableau 6.2 Publication et méthodes de recherche mises en œuvre par les travaux portant sur la publication, recherche et sélection de services Web.

Parmi les travaux étudiés, deux solutions ne proposent pas aux concepteurs de publier librement leurs services. *QWSDataset* et les travaux de [Balke *et al.,* 2003b] se basent sur la découverte de services, par conséquent, ils ne proposent pas de processus de publication classique. En effet, le processus de découverte ne demande pas une étape de publication préalable étant donné que les systèmes découvrent par eux-mêmes les services sur le Web. Tous les autres travaux étudiés proposent aux fournisseurs de publier leurs services Web par l'intermédiaire de solutions qui reposent sur des formalismes et représentation de services Web spécifiques. Nous pensons que dans le cadre de notre travail (génération d'applications adaptées au contexte d'utilisation), l'utilisation unique de la technique de découverte de services n'est pas suffisante. Il faudrait la combiner avec un processus d'enrichissement de la représentation. Ainsi, le fait de publier les services au sein d'un registre public permet : (1) de connaître l'ensemble du corpus des services ; (2) d'enrichir la représentation standard des services faite en WSDL via des catégories d'information, propres à chaque travail. Ceci facilite les processus de recherche et de sélection.

Chacun de ces travaux (excepté le site *XMethods*) propose des méthodes de recherche. Les registres publics sur le Web proposent des méthodes de recherche simples telles que des **recherches par mots-clés** (UDDI et *QWSDataset*), ou proposent aux clients de rechercher les services via une **navigation par catégories** (*RemoteMethods*). Les solutions plus complexes de recherche de services Web sémantiques mettent en œuvre des méthodes qui reposent sur des outils formels. [Balke *et al.,* 2003b] recherchent les services Web par l'intermédiaire d'**alignements d'ontologies**. [Heß *et al.*¸2004], après avoir **réalisé des *clusters* de services équivalents**, les proposent aux clients. [Baader *et al.,* 2005] décrivent les services Web via une logique de description et recherchent ainsi des services par l'intermédiaire des mécanismes d'**inférence** associés. Enfin, [Cabral *et al.*, 2006] utilisent une méthode **de résolution de problèmes** pour rechercher les services Web.

Afin de faciliter l'implémentation d'applications à base de services Web, il est important de proposer tout d'abord aux fournisseurs un moyen de publier leurs services Web pour que ces derniers soient visibles et, ensuite, aux clients, des méthodes de recherche efficaces reposant sur des outils formels. Les travaux qui proposent ces deux aspects, soit offrent des méthodes de recherche simples – par mots-clés (UDDI) ou par navigation par catégories (*RemoteMethods*), soit ne possèdent pas d'interface de communication directe avec leur plate-forme (tels que [Heß *et al.*¸2004], [Baader *et al.,*

2005] et [Cabral *et al.*, 2006]) ce qui ne facilite pas l'utilisation de ces registres par des clients humains.

6.3 Mise en œuvre de la composition de services Web

Les applications à base de services Web résultent de l'invocation d'un ensemble de services. Les étapes de planification et de surveillance de la phase d'assemblage du cycle de vie d'applications à base d'architecture orientée services, permettent de réaliser cette composition de services. Afin de mettre en œuvre ces étapes, il existe des plates-formes de composition, étudiées dans le chapitre 4.

L'**étape de planification** a pour but de définir la composition de services Web. Cette définition doit reposer sur l'utilisation d'un (ou plusieurs) **outil(s) formel(s)** qui permet(tent) d'appliquer des mécanismes de raisonnement qui ont fait leur preuve. Afin de permettre la publication de compositions par les concepteurs au sein des plates-formes, ces dernières doivent proposer l'utilisation de **langages de composition** tels que BPEL4WS, OWL-S, et OCML. De plus, les plates-formes de composition de services Web ont pour rôle de répondre aux problèmes non résolus par les langages de composition, tels que permettre l'**évolutivité de la composition**. Cette évolutivité de la composition repose sur le fait qu'une requête du client peut être satisfaite par un ensemble de compositions équivalentes. Ainsi, avant l'exécution, la plate-forme choisit la définition de la composition qui permet de réduire le nombre de problèmes rencontrés, tels que l'indisponibilité de services.

Concernant l'**étape de surveillance**, son rôle est de prendre en charge le **contrôle de l'exécution** afin d'atteindre l'objectif du client, ainsi que la **compensation de services** si un des services élémentaires n'est pas disponible ou défaillant.

Le Tableau 6.3 permet d'illustrer pour chaque plate-forme d'exécution de composition de services Web, les aspects qu'elle traite durant les étapes de planification et de surveillance du cycle de vie d'applications orientées services.

		SELF-SERV [Sheng *et al.*, 2002]	METEOR-S [Aggarwal *et al.*, 2004]	SHOP2 [Sirin *et al.*, 2004]	IRS-III [Cabral *et al.*, 2006]
Étape de planification	Outil(s) formel(s)	- Diagramme d'états transitions - *Workflows*	- Ontologies (OWL-S) - UDDI Sémantique	- Ontologies (OWL-S) - *Hierachical Task Network*	- Ontologies (WSMO) - *Workflows* - Résolution de problème
	Langage de composition	*Non communiqué*	BPEL4WS	BPEL4WS	OCML
	Évolutivité de la composition (oui/non)	non	non	non	non
Étape de surveillance	Compensation de services (oui/non)	oui	oui	non	non
	Contrôle de l'exécution (oui/non)	oui	oui	non	non

Tableau 6.3 Synthèse des méthodes utilisées et solutions apportées par les plates-formes de composition de services Web.

Afin d'évaluer l'étape de planification, nous décrivons pour chaque plate-forme, la ou les approche(s) sous-jacente(s), le langage de composition utilisé, et indiquons si cette définition de composition est évolutive (*cf.* Tableau 6.3).

Les approches pour la définition de la composition de services Web. Les outils formels utilisés dans les plates-formes diffèrent, bien que toutes aient le même objectif : définir et

exécuter une composition de services Web. La plate-forme SELF-SERV est la seule à ne pas proposer une dimension Web sémantique. Elle définit la composition via des **diagrammes d'états** UML et des ***workflows***. La plate-forme METEOR-S définit les services via le **langage OWL-S** et recherche et sélectionne les services Web élémentaires via une **version améliorée de UDDI**. La plate-forme SHOP2 utilise aussi **OWL-S** pour décrire les services Web et une méthode issue de l'intelligence artificielle (**HTN**) pour définir la composition de services Web. Dans le projet IRS-III, les auteurs ont choisi d'utiliser leur propre ontologie de description de services Web (**WSMO**) et de mettre en œuvre un moteur d'exécution hybride proposant à la fois une composition de type orchestration (avec les ***workflows***) et de type chorégraphie (avec des méthodes de **résolution de problème**).

Le langage de composition. BPEL4WS est le langage utilisé dans les plates-formes METEOR-S et SHOP2. Notons que, bien que OWL-S soit considéré par la communauté des services Web comme un langage de composition de services [Ankolekar *et al.*, 2004], il n'est pas utilisé au sein de ces plates-formes comme tel mais comme langage de description sémantique. La définition de la composition dans IRS-III repose sur le langage OCML. Cependant, ce langage a peu de visibilité étant donné qu'il a été réalisé par les auteurs de IRS-III eux-mêmes. Les auteurs de SELF-SERV [Sheng *et al.*, 2002] ne précisent pas le langage de composition utilisé dans leurs travaux.

L'évolutivité de la composition. Les plates-formes étudiées ne proposent pas d'évolutivité de la composition comme nous l'entendons, c'est-à-dire la possibilité de décrire une composition par le biais de plusieurs définitions afin de choisir la meilleure dans une situation donnée.

Le Tableau 6.3 illustre aussi, pour chaque plate-forme étudiée, si ces dernières prennent en compte les aspects liés à l'étape de surveillance.

Le contrôle de l'exécution. Les architectures des plates-formes SELF-SERV et METEOR-S possèdent toutes les deux un composant qui se charge de la composition de services Web et en contrôle l'exécution (le composant *Coordinator* pour SELF-SERV et le composant *Binder* pour METEOR-S).

La compensation de services. La plate-forme SELF-SERV propose un mécanisme de compensation par le biais des communautés de services qui regroupent un ensemble de services Web qui répondent au même objectif. Lors de l'exécution de la composition, si un service échoue, la plate-forme METEOR-S réitère le processus de recherche et sélection de services.

D'après l'étude de ces plates-formes d'exécution de la composition de services Web, nous mettons en évidence trois points : (1) il est nécessaire de définir une composition de services Web à l'aide d'outils formels existants afin d'utiliser leurs mécanismes de raisonnement et de faciliter la tâche des concepteurs lors de la définition de la composition ; (2) l'évolutivité de la composition est un paradigme non pris en compte aujourd'hui ; (3) la surveillance de la composition doit satisfaire la compensation de services et le contrôle de l'exécution. Aucune plate-forme étudiée ici ne répond à l'ensemble de ces points concernant la composition de services Web.

6.4 Mise en œuvre de l'adaptation dans le domaine des services Web

Dans le chapitre 5, nous avons recensé des solutions relatives à la mise en œuvre de l'adaptation dans le domaine des services Web. Ces dernières reposent généralement sur un enrichissement de la représentation des services eux-mêmes et de la requête du client. Nous comparons ces solutions dans le paragraphe 6.4.1. Cette étude nous a permis d'identifier trois approches d'adaptation (l'adaptation du résultat, de la demande de services, et de la composition) et trois techniques d'adaptation (par conversion, par sélection de services Web, et par sélection de compositions). Dans le paragraphe 6.4.2

nous rappelons les aspects liés à ces formes et techniques d'adaptation. Nous concluons cette section par l'évaluation des techniques d'adaptation préalablement définies (*cf.* paragraphe 6.4.3).

6.4.1 Représentation enrichie des services Web et des requêtes

La mise en œuvre de l'adaptation au contexte dans le domaine des services Web (élémentaires ou composants) repose sur deux points : la définition du contexte pour lequel il existe une forme d'adaptation, et l'enrichissement de la représentation, soit du service, soit de la requête du client, soit des deux par l'intermédiaire du contexte préalablement défini.

Nous avons déterminé quatre aspects pris en compte dans la **définition du contexte** dans le domaine des services Web : l'utilisateur, les aspects liés à la mobilité, la présentation du contenu, et la donnée.

L'utilisateur. Cette catégorie d'information permet de décrire tant les informations personnelles de l'utilisateur (telles que son nom, son adresse, ses préférences, ses intérêts, ses activités), que les informations liées à l'utilisation de l'application (telles que les préférences de connexion et d'utilisation de l'utilisateur ou ses tâches). Les informations précises concernant l'utilisateur diffèrent d'un travail à un autre.

Les aspects liés à la mobilité. Nous avons vu, dans le chapitre 5, que les domaines de l'informatique ubiquitaire, pervasive, et sensible au contexte, sont des domaines qui traitent de l'adaptation au contexte. Une des problématiques de ces domaines est l'utilisation des applications dans un contexte de mobilité des utilisateurs et des ressources. De ce fait, il est naturel que la définition du contexte englobe les aspects liés à la mobilité. D'après les études réalisées, nous déterminons trois catégories d'information liées à la mobilité de l'utilisateur : la **localisation** de l'utilisateur, le **dispositif** utilisé (la taille de l'écran, les caractéristiques matérielles et logicielles du dispositif, etc.) et les informations sur l'**environnement** (la date, l'heure, la lumière, le bruit environnant, les personnes se trouvant à proximité de l'utilisateur).

La présentation du contenu. Afin que le résultat de l'appel du service soit lisible sur le dispositif de l'utilisateur final, quel que soit son type de terminal, il faut prendre en compte au sein du contexte la présentation du contenu.

Le contexte de la donnée. Lors de la composition de services, la sortie d'un service devient l'entrée du suivant. Afin que ces services puissent communiquer, il faut que le contexte de la donnée (structure et type de données) de sortie du premier service corresponde au contexte de la donnée de l'entrée du second service. Il est donc primordial de prendre en compte le contexte de la donnée afin de garantir la communication entre les services.

Ces catégories d'information qui définissent le contexte permettent d'enrichir la représentation du service et de la requête.

Représentation enrichie de la requête. Lors de l'intégration du contexte dans la représentation de la requête, les systèmes peuvent alors connaître le contexte pour lequel le client cherche une forme d'adaptation.

Représentation enrichie des services. Lorsque l'on enrichit la représentation du service à l'aide de la définition du contexte, les clients peuvent connaître le contexte pour lequel le service est adapté.

Représentations enrichies des services et de la requête. La mise en correspondance entre la requête et les services disponibles selon un contexte peut reposer sur l'intégration du contexte dans les représentations de ces deux entités.

La formalisation de ces représentations repose, soit sur des langages semi-structurés, tels que XML, soit sur des langages issus du Web sémantique. Ces derniers apportent des avantages en termes d'interprétation avec, par exemple, l'utilisation ultérieure de mécanismes d'inférence, d'alignement de descriptions, etc.

Le Tableau 6.4 illustre les catégories d'information prises en compte dans les définitions du contexte pour chaque travail.

L'utilisateur. Les informations concernant l'utilisateur sont prises en compte par tous les travaux étudiés, excepté [Mrissa, 2007].

Les aspects liés à la mobilité. Tous les travaux, exceptés [Mrissa, 2007] (qui ne se préoccupe que du contexte de la donnée) et [Balke *et al.,* 2003b] (qui ne se préoccupent que de l'utilisateur), intègrent les aspects liés à la mobilité à la définition du contexte qu'ils utilisent. Seuls [Pashtan *et al*., 2004] incluent les trois catégories d'information dans la définition du contexte.

La présentation du contenu. Seuls [Keidl *et al.,* 2004] intègrent dans la définition du contexte la présentation du contenu en vue de l'adapter.

Le contexte de la donnée. [Mrissa, 2007] n'inclut que le contexte de la donnée dans la définition du contexte. Ce travail est le seul à prendre en compte cette catégorie d'information. Ce travail est intéressant en termes de méthodes utilisées pour mettre en œuvre l'adaptation, mais ne se préoccupe pas des aspects d'adaptation relatifs aux domaines de l'informatique ubiquitaire ou pervasive.

		Utilisateur	Aspects liés à la mobilité			Présentation du résultat	Contexte de la donnée
			Localisation	Dispositif	Environnement		
Services Web élémentaires	Services Web centrés utilisateur [Balke *et al.*, 2003b]	S+ / R+	-	-	-	-	-
	Adaptation de l'exécution de services [Keidl *et al.*, 2004]	R	R	R	-	R	-
	Services Web mobiles [Pashtan *et al.*, 2004]	R	R	R	R	-	-
	Services Web sensibles au contexte [Suraci *et al.*, 2007]	S+ / R+	S+ / R+	-	S+ / R+	-	-
Composition de services Web	Annotation de WSDL et médiation [Mrissa, 2007]	-	-	-	-	-	S+
	Composition orientée contexte [Vukovic *et al.*, 2004]	R	-	R	R	-	-
	Approche par automates d'états finis [Ben Mokhtar *et al.*, 2005b]	S+ / R+	-	S+ / R+	S+ / R+	-	-
	Approche par planification [Qiu *et al.*, 2007]	S+ / R+	S+ / R+	-	S+ / R+	-	-

Légende :
'**S**' désigne que la **description du service** comporte cette catégorie d'information
'**R**' désigne que la **description de la requête** comporte cette catégorie d'information
'**+**' désigne que le travail utilise des **langages du Web sémantique**
'-' désigne que la catégorie d'information n'est pas prise en compte

Tableau 6.4 Catégories d'information du contexte enrichissant les représentations des services Web et de la requête du client, dans les travaux d'adaptation de services Web élémentaires ou composants.

Le Tableau 6.4 indique si le contexte défini est inclus dans la représentation du service (S) de la requête (R) ou des deux (S/R).

Représentation enrichie de la requête. Tous les travaux, excepté celui de [Mrissa, 2007] enrichissent la représentation de la requête des services à l'aide du contexte. Nous pensons que [Mrissa, 2007] ne propose pas cette solution étant donné que le contexte qu'il a défini (celui de la donnée) est automatiquement et de manière implicite pris en compte dans l'invocation du

service. [Keidl *et al.,* 2004] [Pashtan *et al.,* 2004] [Vukovic *et al.,* 2004] ne propose qu'un enrichissement de la représentation de la requête.

Représentation enrichie des services. Seul [Mrissa, 2007] propose d'enrichir uniquement la requête des services. L'intégration du contexte dans la représentation du service est réalisée par des annotations sémantiques de WSDL.

Représentations enrichies des services et de la requête. [Balke *et al.,* 2003b] et [Suraci *et al.*, 2007] ont choisi d'utiliser des ontologies qui leurs sont spécifiques, tandis que [Ben Mokhtar *et al.*, 2005b] et [Qiu *et al.*, 2006] ont choisi d'étendre le langage OWL-S.

De notre point de vue, l'enrichissement de la requête par la définition du contexte pour lequel le client recherche une forme d'adaptation est indispensable. Concernant la représentation du service, nous pensons que l'intégration du contexte est nécessaire seulement si le service Web propose un résultat adapté à un contexte particulier.

6.4.2 Approches et techniques d'adaptation

Lors de la définition de la problématique que nous abordons dans cette thèse, nous avons mis en évidence les phases et étapes du cycle de vie où l'adaptation peut être mise en œuvre. En conclusion du chapitre sur l'adaptation au contexte dans le domaine des services Web (chapitre 5), différentes formes d'adaptation ont été identifiées. Dans cette sous-section, nous faisons le lien entre les **formes d'adaptation** identifiées (adaptation du résultat du service, de la demande de services et de la composition), les **techniques d'adaptation** utilisées (conversion du résultat, sélection de services Web et sélection de compositions) et les **phases et étapes du cycle de vie** des applications ayant une architecture orientée services.

Adaptation du résultat du service. L'adaptation du contenu que nous avons identifié dans les travaux étudiés (ceux de [Pashtan *et al*., 2004] [Mrissa, 2007]), consiste à **convertir les résultats** du (ou des) service(s) invoqués afin qu'ils conviennent au contexte d'utilisation. Cette forme d'adaptation est établie lors de la phase d'**assemblage** du cycle de vie des applications à architecture orientée services. Lors de l'étape de **planification**, les clients mettent en œuvre l'adaptation du contenu en faisant appel à un composant (service Web ou non) afin que le résultat soit adapté au contexte d'utilisation. Lors de l'étape de **surveillance**, si le résultat final ne convient pas au contexte d'utilisation, le client peut aussi, de la même manière, mettre en œuvre cette forme d'adaptation.

Adaptation de la demande de services. Dans la phase de **conception**, les clients définissent leurs besoins afin d'implémenter l'application. Si leur objectif est de mettre en œuvre une application adaptée, les concepteurs doivent prendre en compte cette dimension lors des étapes de **recherche** et **sélection** de services Web. Certains travaux proposent aux clients des solutions afin de trouver les services Web qui conviennent le mieux à **la demande** du client selon un contexte donné par **sélection** de services Web.

Adaptation de la composition. Une dernière forme d'adaptation dans le domaine des services Web est l'adaptation de la **composition**. Cette adaptation est réalisée dans la phase d'**assemblage** du cycle de vie de l'application et peut pendre deux formes. Tout d'abord, elle peut se dérouler dans l'étape de **planification**. Le travail propose alors différentes stratégies de composition par le biais de la technique d'adaptation par **sélection de compositions**. Ensuite l'adaptation de la composition peut se situer dans l'étape de **surveillance** et consiste à trouver un autre service Web si celui appelé en premier lieu ne correspond pas au contexte d'utilisation. Ceci consiste à utiliser la technique d'adaptation par **sélection de services Web**.

Le Tableau 6.5 illustre ces liens et les formes d'adaptation réalisées par les travaux étudiés dans le chapitre 5, portant sur la mise en œuvre de l'adaptation dans le domaine des services Web.

		Forme d'adaptation		
		Adaptation du résultat	Adaptation de la demande de services	Adaptation de la composition
Services Web élémentaires	Services Web centrés utilisateur [Balke *et al.*, 2003b]		- Sélection de SW	
	Adaptation de l'exécution de services [Keidl *et al.*, 2004]		- Sélection de SW	
	Services Web mobiles [Pashtan *et al.*, 2004]	- Conversion du résultat		
	Services Web sensibles au contexte [Suraci *et al.*, 2007]		- Sélection de SW	
Composition de services Web	Annotation de WSDL et médiation [Mrissa, 2007]	- Conversion du résultat		
	Composition orientée contexte [Vukovic *et al.*, 2004]			- Sélection de SW
	Approche par automates d'états finis [Ben Mokhtar *et al.*, 2005b]			- Sélection de SW
	Approche par planification [Qiu *et al.*, 2007]			- Sélection de SW - Sélection de compositions

Tableau 6.5 Synthèse des travaux portant sur l'adaptation des services Web en soulignant les formes et techniques d'adaptation mises en œuvre.

Parmi les travaux étudiés, chacun se préoccupe d'une forme d'adaptation particulière et ne met en œuvre qu'une technique d'adaptation, excepté [Qiu *et al.*, 2007] qui en mettent en œuvre deux (sélection de services Web et d'une composition). Nous pensons qu'un travail qui permet aux concepteurs de réaliser des applications adaptées au contexte, doit prendre en compte l'adaptation tout au long du cycle de vie de l'application et ainsi proposer toutes les formes (de contenu, de demande de services Web et de la composition) et techniques d'adaptation (par conversion, sélection de services Web et d'une composition).

6.4.3 Évaluation des techniques d'adaptation

Dans le domaine des hypermédia adaptatifs, il existe des travaux qui déterminent des critères d'évaluation pour les capacités d'adaptation des Systèmes d'Information sur le Web (SIW). Nous rapprochons ici ces critères aux techniques d'adaptation des services Web que nous avons définies préalablement (par conversion, par sélection de services Web et par sélection de compositions).

[Villanova-Oliver, 2002] définit trois critères d'évaluation relatifs aux capacités d'adaptation des SIW : l'adaptation minimale, l'adaptabilité et l'adaptativité. Nous avons modifié ces définitions en les rapprochant des systèmes à base de services Web.

L'adaptation minimale. Ce critère définit si les systèmes permettent de choisir un service Web parmi un ensemble de services Web défini au préalable.

L'adaptabilité. Ce critère définit l'aptitude des systèmes à réagir à des demandes explicites d'adaptation de la part de l'utilisateur.

L'adaptativité. Ce critère définit l'aptitude des systèmes à observer le comportement de l'utilisateur (ou dans notre cas des services Web composant le système), et à procéder en conséquence à des adaptations.

La Figure 6.1 illustre les critères d'évaluation des trois techniques d'adaptation relevées dans le domaine des services Web. Ces critères sont classés selon deux catégories : le moment de la réalisation de l'adaptation et le contrôle de l'adaptation.

Le moment de la réalisation de l'adaptation. [Stephanidis *et al.*, 1998] [Frasincar *et al.*, 2002] [Kappel *et al.*, 2000] distinguent les capacités d'adaptation selon le moment de sa réalisation. Dans les définitions originelles, les auteurs identifient deux moments de la réalisation de l'adaptation : le temps réel (*i.e.* le système met en œuvre l'adaptation durant son exécution) et l'initialisation (*i.e.* la mise en œuvre de l'adaptation est réalisée avant l'exécution du système). Dans le cadre de notre travail, nous séparons le temps réel, mis en œuvre par l'étape de **surveillance** du cycle des systèmes à base de services Web, et le moment de la conception (correspondant à la désignation « initialisation » dans les définitions originelles), mis en œuvre par les étapes de **recherche**, **sélection** et **planification** du cycle de vie.

Le contrôle de l'adaptation. [Dietrich *et al.*, 1993] [Oppermann, 1994] [Kobsa *et al.*, 2001] distinguent les capacités d'adaptation selon l'acteur qui réalise le contrôle de l'adaptation. Dans les définitions originelles, les auteurs séparent le contrôle par l'utilisateur et le contrôle par le système. Dans le cadre de notre travail, nous discernons le contrôle par un acteur humain (qui peut être, soit le client qui conçoit le système, soit l'utilisateur final) et par un acteur logiciel (qui peut être le système lui-même ou un acteur logiciel tel qu'un service Web).

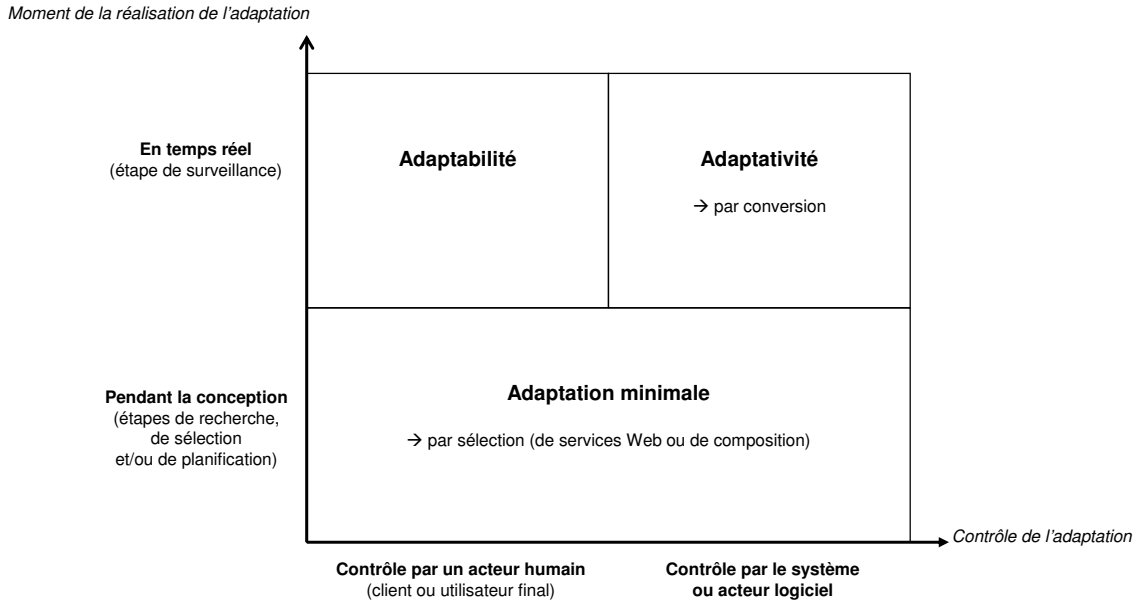


Figure 6.1 Typologie retenue pour l'état de l'art des techniques d'adaptation dans le domaine des services Web (inspirée de [Villanova-Oliver, 2002]).

Les travaux étudiés dans l'état de l'art proposent en majorité une adaptation minimale à l'aide de sélection de services Web [Balke *et al.,* 2003a] [Keidl *et al.,* 2004] [Suraci *et al.*, 2007] [Vukovic *et al.*, 2004] [Ben Mokhtar *et al.*, 2005b] [Qiu *et al.*, 2007] ou de composition [Qiu *et al.*, 2007]. Seuls [Pashtan *et al.*, 2004] [Mrissa, 2007] proposent de l'adaptativité en mettant en œuvre l'adaptation par le biais de la technique d'adaptation par conversion. Aucun de ces travaux ne propose une adaptation de type adaptabilité.

6.5 Objectifs de notre proposition

La synthèse de l'état de l'art a mis en évidence les apports et les manques des différents travaux issus du domaine des services Web portant sur les étapes du cycle de vie d'une application à base de services Web :

la **description** et la **publication** des services par le fournisseur ;

la **recherche** et la **sélection** des services Web convenant le mieux à la demande du client ;

la **composition** des services Web en vue d'implémenter une application.

En plus du fait que l'adaptation s'applique à différentes étapes du cycle de vie, nous avons souligné les travaux mettant en œuvre de l'**adaptation** au sein des applications à base de services Web.

Aucun travail étudié dans l'état de l'art ne prend en compte l'ensemble du cycle de vie d'une application orientée en y intégrant la mise en œuvre de l'adaptation. Seuls le projet IRS-III [Cabral *et al.*, 2006] et les travaux de [Balke *et al.,* 2003b] couvrent presque l'intégralité du cycle de vie. Le projet IRS-III [Cabral *et al.*, 2006] englobe les étapes de description, de publication, de recherche, de sélection et de composition de services Web, mais ne considère pas l'adaptation. [Balke *et al.,* 2003b] traitent une approche visant la description, la recherche et la sélection de services Web en intégrant l'adaptation, mais ne s'intéressent pas à la composition.

L'objectif de cette thèse est de proposer aux concepteurs d'applications une plate-forme pour générer des applications à base de services Web adaptées au contexte. Notre proposition traite de manière transversale le cycle de vie d'une telle application et propose d'intégrer à chacune de ces étapes une forme et une technique d'adaptation. Notre proposition vise trois objectifs entrelacés en vue de mettre en œuvre une plate-forme de génération d'applications adaptées au contexte à base de services Web : (i) la publication, la recherche et la sélection des services ; (ii) la composition des services Web ; et (iii) la prise en compte de l'adaptation. La Figure 6.2 illustre les dépendances entre les objectifs (représentés ici par la (ou les) étape(s) du cycle où ils interviennent) et leurs réalisations (par des modèles et des implémentations).

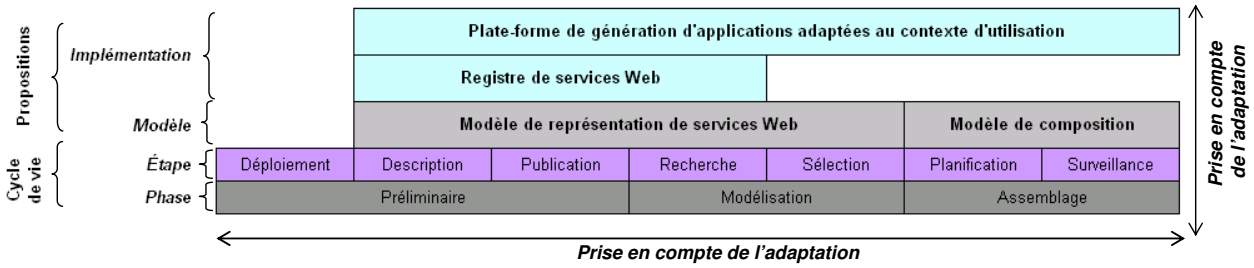


Figure 6.2 Illustration des moyens mis en œuvre pour atteindre nos objectifs.

L'objectif concernant la publication, la recherche, et la sélection. Afin de rendre plus efficaces les étapes de recherche et de sélection, l'étape de publication intègre une représentation enrichie des services Web (supportée par le **modèle de représentation de services Web**). La requête de recherche et les mécanismes de sélection de services doivent de même être enrichis afin de trouver les services Web adaptés aux attentes du client. Notre proposition de **registre de services Web**, intégrant des méthodes de recherche, vise à faciliter les tâches des fournisseurs et des clients lors de ces trois étapes du cycle de vie d'une application à base d'AOS.

L'objectif concernant la composition. Aujourd'hui, les plates-formes de composition n'offrent pas de moyen de rendre évolutive la définition de la composition (étape de *Planification*), ni une possibilité de contrôler l'exécution de la composition et de compenser les services défaillants (étape de *Surveillance*). Dans ce sens, nous proposons un **modèle de composition** qui repose sur

des langages issus du domaine de l'intelligence artificielle. Ainsi ce modèle est enrichi des apports spécifiques à ce domaine et met en œuvre des solutions en termes d'évolutivité, de surveillance de la composition et de compensation de services.

L'objectif concernant l'adaptation. La mise en œuvre de l'adaptation dans le domaine des services Web repose sur trois types d'adaptation (adaptation du résultat, de la demande de services Web et de la composition) et trois techniques d'adaptation (conversion, sélection de services Web et sélection de compositions). Nous utilisons ces trois types d'adaptation et ces trois techniques d'adaptation au sein de notre proposition. De plus, nous proposons deux nouvelles techniques d'adaptation afin que le processus d'adaptation soit mis en œuvre à tous les niveaux du cycle de vie de l'application : de la *phase préliminaire* à la *phase d'assemblage*. Ces deux nouvelles techniques d'adaptation sont : *la mise en œuvre de services d'adaptation*, qui consiste à déployer des services spécifiques à l'adaptation, et *la compensation de services*, qui consiste à trouver un service équivalent lors de la défaillance d'un service.

L'ensemble de ces objectifs est intégré à une **plate-forme de génération d'applications adaptées au contexte d'utilisation à base de services Web**.

La seconde partie de ce mémoire de thèse décrit notre proposition qui répond aux objectifs cités plus haut. Cette partie est décomposée en cinq sections, portant sur : le modèle de représentation de services Web, le modèle de composition de services Web, l'opérationnalisation de ces modèles, le registre de services Web, et la plate-forme de génération d'applications adaptées basées sur des services Web.

PROPOSITION

7 MODÈLE DE REPRÉSENTATION DE SERVICES WEB

L'un des objectifs de notre travail est de permettre aux fournisseurs de rendre visibles leurs services Web et de faciliter, pour les clients, la mise en œuvre d'application à base d'architecture orientée services Web. Les premières étapes sous-jacentes à la mise en œuvre d'une telle application sont **la publication de services Web** développés par un fournisseur, **la recherche et la sélection de services Web** existants réalisées par les clients. Nous considérons que les étapes de description et d'invocation ont atteint un degré de maturité important en s'appuyant sur les standards WSDL et SOAP.

Il existe de nombreux registres sur le Web, ou des travaux de recherche, qui proposent un moyen de publier, de rechercher et de sélectionner les services Web (*cf.* chapitre 3). Cependant, les méthodes de recherche (par exemple par mots-clés) et la représentation des services ne suffisent pas à répondre à l'attente précise du client lors de la conception d'applications. L'étude faite des registres existants nous a permis de mettre en évidence quatre catégories d'information qui, de notre point de vue, sont indispensables à prendre en compte dans la représentation de services Web, en vue de faciliter la recherche et la sélection de services Web : le service lui-même, le fournisseur du service et l'information associée, le domaine d'application et le contexte d'utilisation.

Nous proposons un modèle de représentation de services [Lopez-Velasco *et al.*, 2007a], nommé WSR-Model (*Web Service Representation Model*), qui enrichit la description d'un service en y intégrant les quatre catégories d'information que nous jugeons pertinentes à prendre en compte. Nous définissons la représentation de services Web comme un ensemble de catégories d'information qui caractérise un service Web lors des étapes de description et de publication. Le renseignement des informations relatives à la description d'un service est à la charge du fournisseur du service. Si le service publié est issu de la technologie des services Web, une partie des informations sous-jacentes à la description du service lui-même est extraite automatiquement de sa description WSDL afin d'alléger la charge de travail du fournisseur. Du point de vue client, WSR-Model permet de formaliser la requête de recherche de services. La représentation des services Web renvoyés par une recherche facilite la tâche de sélection, réalisée par les clients.

Dans ce chapitre, nous décrivons tout d'abord les besoins en termes d'information en vue de réaliser le modèle de représentation de services Web. La seconde section est consacrée à la

présentation du modèle du point de vue fournisseur. Ensuite, nous déterminons en quoi WSR-Model permet de caractériser les requêtes des clients en termes de recherche de services. Enfin, nous mettons en évidence les approches d'adaptation qui sont permises par la suite via l'utilisation de ce modèle.

7.1 Identification des besoins en termes de représentation de services

Cette section met en relief les quatre catégories d'information dont les clients ont besoin afin de mieux rechercher et sélectionner les services Web. Nous proposons que ces catégories d'information enrichissent la représentation standard WSDL d'un service Web. Nous considérons les catégories suivantes : le service Web, son fournisseur, le domaine d'application pour lequel il a été développé, et le contexte d'utilisation pour lequel le service peut apporter une réponse adéquate.

Le service. Afin de sélectionner un service, il est important de connaître ses aspects fonctionnels. Cette catégorie d'information rassemble les méthodes proposées, les paramètres d'entrée et de sortie, le protocole à utiliser pour appeler le service, et la localisation du service (son URL). WSDL permet de décrire ce type d'information fonctionnelle.

Le fournisseur. Cette catégorie rassemble tant la description du fournisseur que des informations associées à l'utilisation du service. Nous pensons que la description du fournisseur est une information que le client doit connaître pour deux raisons.

- Cette description peut orienter et faciliter le choix du service lors du processus de sélection. En effet, la confiance accordée au fournisseur, le coût (en termes de droit d'invocation), le temps de réponse du service, la confidentialité des informations transmises, etc., sont des informations qui peuvent intervenir dans les critères de sélection.
- La description du fournisseur est intéressante à prendre en compte pour la maintenance du service. Si, lors de l'exécution du service, un concepteur fait face à des problèmes (par exemple, un problème de connexion), si ce dernier est en possession d'informations sur le fournisseur, il peut alors prendre contact avec le responsable du service pour une assistance technique. De plus, le client peut prendre contact avec le fournisseur à la suite de l'utilisation de ses services et ainsi lui transmettre des informations sur l'utilisabilité des services telles que des suggestions d'amélioration d'utilisation du service.

Le domaine d'application. Une application couvre un ensemble de fonctionnalités qui peuvent être exécutées par des services Web qui ciblent des données et des opérations spécifiques à, en général, un domaine d'application, tel que la géomatique, le *business-to-business* ou le *e-learning*. Si nous nous intéressons à la conception de telles applications, l'ensemble des fonctionnalités mises en jeu sont autant de services Web à invoquer (par conséquent à rechercher, à sélectionner, et si besoin est, à développer). Le fait d'associer à la représentation du service le domaine d'application dans lequel il peut intervenir et les fonctionnalités qu'il met en œuvre permet de faciliter la recherche et la sélection de services.

Le contexte d'utilisation. Les fournisseurs de services Web doivent prendre en compte le plus tôt possible lors de la conception de systèmes le besoin d'adaptation au contexte afin de répondre au mieux aux attentes des utilisateurs. Si les fournisseurs de services Web mettent à la disposition des clients une description du contexte d'utilisation pour lequel le service est adapté, la recherche et la sélection peuvent être facilitées. Dans notre travail, nous nous focalisons sur les éléments du contexte qui caractérisent l'utilisation d'un système, pouvant être détectés et utilisés par ce système afin d'offrir un résultat adapté. Nous appelons cet ensemble d'éléments le contexte d'utilisation. Nous considérons quatre principaux éléments dans la description du

contexte d'utilisation : l'utilisateur (droits d'accès, activité, etc.), l'information liée à la localisation (coordonnées GPS, température, etc.), le temps (heure, jour, mois, etc.) et les entités informatiques (caractéristiques logicielles et matérielles du dispositif, le réseau utilisé, etc.).

Les catégories d'information que nous venons d'identifier (service, fournisseur, domaine d'application et contexte d'utilisation) sont intégrées au sein de la représentation des services Web par l'intermédiaire du modèle que nous décrivons dans la section suivante.

7.2 WSR-Model du point de vue du fournisseur

Dans cette section, nous décrivons le modèle de représentation de services Web que nous proposons du point de vue du fournisseur. Nous décrivons exclusivement les packages, les classes, et les relations utilisés dans le cadre de l'activité d'un fournisseur, c'est-à-dire le processus de publication. Dans un premier temps, nous donnons une vue d'ensemble du modèle, puis, nous détaillons chacune des parties le composant. Enfin, nous définissons les liens qui existent entre chaque partie du modèle.

7.2.1 Vue d'ensemble du modèle de représentation de services

Le modèle de représentation de services Web enrichit la description de services afin de faciliter les phases ultérieures de recherche et de sélection réalisées par le client. Cinq packages UML [Booch *et al.*, 1998], chacun correspondant à un critère de notre représentation de services Web, composent le modèle (*cf.* Figure 7.1) : *Service*, *Application Domain*, *Functional Profile*, *Non Functional Profile*, *Use Context*.

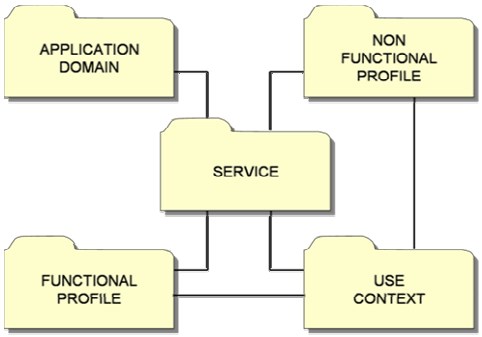


Figure 7.1 Vue d'ensemble du modèle de représentation de services Web.

Service. Ce package représente le service décrit. Il est le package central du modèle et est relié à tous les autres packages.

Application Domain. Le package *Application Domain* permet de décrire le domaine associé aux services représentés.

Functional Profile. Toutes les informations fonctionnelles permettant aux concepteurs d'utiliser le service sont intégrées dans le package nommé *Functional Profile*.

Non Functional Profile. Ce package rassemble les informations à propos du service, non directement liées à son utilisation mais utiles lors du processus de sélection (telles que la description du fournisseur, les conditions d'utilisation, etc.).

Use Context. Ce package intègre la description du contexte d'utilisation dans notre modèle afin de représenter le contexte d'utilisation auquel le service publié est adapté.

Le Tableau 7.1 illustre les associations entre les catégories d'information, déterminées dans la section précédente, qui enrichissent la représentation des services Web, et les packages du modèle.

Catégories d'information identifiées	Package(s) correspondant aux catégories dans le modèle de représentation de services Web
Service	Service Functional Profile Non Functional Profile
Domaine d'application	Application Domain
Fournisseur	Non Functional Profile
Contexte d'utilisation	Use Context

Tableau 7.1 Packages du modèle de représentation de services Web associés aux catégories d'information identifiées précédemment.

Nous avons choisi de décomposer la catégorie représentant le service en trois packages distincts (*Service*, *Functional Profile*, *Non Functional Profile*) étant donné que chacun d'eux est utilisé de manière différente :

- le package *Service* permet de décrire de manière nominale le service décrit et établit le lien entre l'ensemble des packages.
- Le package *Functional Profile* regroupe les informations principalement utilisées lors de l'étape de sélection pour faciliter la tâche du client.
- Le package *Non Functional Profile* est principalement utilisé lors de l'étape d'invocation du service et est directement instancié par la description WSDL, lorsque le service décrit est un service Web.

Dans ce modèle de représentation de services, chacun des cinq packages peut être utilisé indépendamment :

Le package *Service* (représentant les informations élémentaires d'un service) décrit tant un programme logiciel classique (tel qu'un composant), qu'un service issu de la technologie des services Web. Par conséquent, notre modèle peut être utilisé pour représenter tout type de composant logiciel, issu de la technologie des services Web ou non.

Le package *Application Domain* est utilisé dans ce modèle pour décrire le domaine d'application et les différentes fonctionnalités associées aux services publiés.

Les packages *Functional Profile* et *Non Functional Profile* peuvent être utilisés pour compléter une description de n'importe quel composant réutilisable.

Le package *Use Context* offre un moyen de formaliser le contexte d'utilisation en vue de décrire le contexte en adéquation avec l'exécution d'un service.

Les sous-sections qui suivent décrivent chacun des packages composant le modèle de représentation de services (*Service*, *Application Domain*, *Functional Profile*, *Non Functional Profile*, *Use Context*). Une dernière partie est consacrée aux liens existants entre les packages.

7.2.2 Représentation du service

Le package *Service* décrit les informations élémentaires à propos du service à travers trois classes (*cf.* Figure 7.2). Les deux premières classes (*ServiceCategory* et *ConcreteService*) représentent n'importe quel type de service (issu de la technologie des services Web ou non), alors que la troisième classe (*WebService*) est spécifique à l'utilisation de services Web.

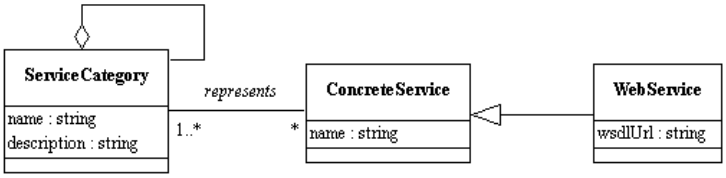


Figure 7.2 Classes et associations du package *Service*.

La classe *ServiceCategory*. À l'aide de cette classe, le modèle fournit un moyen de décrire la catégorie du service, par le biais de deux attributs (*name* et *description*). Par exemple, un service qui propose un moyen de localiser un utilisateur muni d'un dispositif mobile appartient à la catégorie *Localization* (instance de la classe *ServiceCategory*). Nous offrons la possibilité d'exprimer des catégories qui composent une catégorie par le biais de la relation de composition sur cette même classe.

La classe *ConcreteService*. Cette classe représente le service réellement publié par le fournisseur. Un service concret est décrit par son nom (attribut *name*) et par la (ou les) catégorie(s) à laquelle il est associé (association entre les classes *ServiceCategory* et *ConcreteService*). Un service doit appartenir au moins à une catégorie.

La classe *WebService*. Cette troisième classe du package *Service* est celle qui spécifie les services Web. Elle spécialise la classe *ConcreteService* et possède un attribut (*wsdlUrl*) qui permet aux clients du service de localiser la description standard (exprimée en WSDL) du service Web.

Le package *Service* permet aux fournisseurs de services Web de décrire leurs services Web en termes de catégorie de services, ce qui permet de les rendre plus visibles pour les clients à la recherche des services appartenant à une catégorie spécifique.

7.2.3 Représentation du domaine d'application

Le package *Application Domain* permet de caractériser le domaine d'application. L'utilisation de ce package par le fournisseur de services lors de leur publication consiste à ajouter un niveau de description supplémentaire. Ce package est composé de trois classes (*cf.* Figure 7.3) : *Domain*, *Application* et *Functionality*.

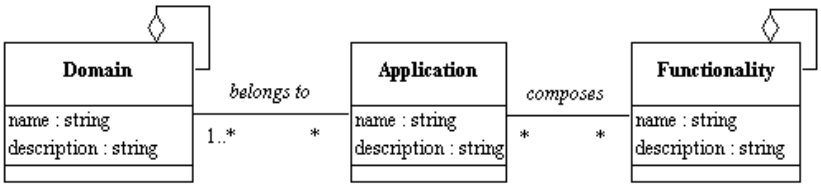


Figure 7.3 Classes et associations du package *Application Domain*.

La classe *Domain*. Cette classe décrit le domaine d'application dans lequel le service intervient. L'instance de cette classe peut être un domaine tel que la géomatique, le travail collaboratif, le *e-learning*, etc. Dans ce package, nous proposons de formaliser le fait qu'un domaine peut être composé de sous-domaines (par exemple le domaine de la géomatique est composé du domaine de la cartographie) par la relation de composition sur la classe *Domain*.

La classe *Application*. Dans ce modèle nous associons à un domaine une ou plusieurs applications (classe *Application*) faisant référence ici à des applications de type logiciel. L'association entre les classes *Domain* et *Application* permet de représenter l'ensemble des applications pouvant intervenir dans un domaine donné. Par exemple, l'application *TomTom*[64] (logiciel de navigation) est une instance de la classe *Application* et est associée à l'instance géomatique de la classe *Domain*.

La classe *Functionality*. Cette classe permet de représenter le fait qu'une application est développée à l'aide d'un ou plusieurs composants qui implémentent chacun une fonctionnalité spécifique. Cette classe décrit les fonctionnalités proposées par l'application. Si une tâche est trop complexe, elle peut être composée de sous-tâche(s) (introduite(s) par la relation de composition sur la classe *Functionality*). Considérons les tâches suivantes : l'affichage de carte, le calcul d'itinéraire, l'affichage de l'itinéraire et la localisation de l'utilisateur. Ces instances de la classe *Functionality* sont associées à l'instance *TomTom* de la classe *Application* à l'aide de l'association entre les classes *Application* et *Functionality*.

L'intérêt du package *Application Domain* est de permettre aux fournisseurs de services d'enrichir leur représentation en décrivant pour quel domaine d'application (classe *Domain*) et application (classe *Application*) leurs services ont été conçus et la fonctionnalité que le service fournit (via la classe *Functionality*).

7.2.4 Représentation du profil fonctionnel

Le package nommé *Functional Profile* rassemble les informations à publier afin que les concepteurs puissent utiliser les services. Nous nous sommes basés sur le modèle conceptuel de WSDL afin de construire ce package. La transcription de fichiers WSDL, exprimés en XML, en représentation orientée objet (telle que UML) a déjà fait l'objet de travaux, tels que ceux de [Marcos *et al.*, 2003]. Cependant, ces travaux sont d'un niveau d'abstraction trop faible et étroitement lié au standard WSDL. Dans notre travail, nous nous concentrons sur une description des éléments de haut niveau tels que les méthodes et les arguments de ces méthodes. De plus, nous avons choisi de proposer un modèle indépendant du type de service publié, par conséquent indépendant de la technologie des services Web. Le package *Functional Profile* est composé de cinq classes (*cf.* Figure 7.4) : *Method*, *Binding*, *Parameter*, *InParameter* et *OutParameter*. Lorsque la publication concerne des services Web, les informations représentées dans ce package sont celles issues de la description WSDL, par conséquent peuvent être instanciées de manière automatique. Dans le cas de la publication de services non issus de la technologie Web, le fournisseur doit renseigner par lui-même les valeurs de ces classes.

La classe *Method*. Cette classe décrit les méthodes proposées par les services décrits.

La classe *Binding*. Lorsque le service est hébergé sur le Web, le fournisseur doit publier le moyen d'accès au service (*i.e.* le protocole à utiliser, tel que SOAP ou HTTP). La classe *Binding* représente ce type d'information. Une instance de cette classe peut être HTTP ou SOAP. Chaque méthode fournie par un même service peut être accessible par un protocole différent. Le package

[64] http://www.tomtom.com/

Functional Profile contient une relation entre la classe *Method* et la classe *Binding* afin que les clients puissent connaître pour chacune des méthodes disponibles leur protocole d'accès.

Les classes *Parameter*, *InParameter* et *OutParameter*. Les paramètres (classe *Parameter*) de chaque méthode sont représentés dans le modèle, par l'association reliant les classes *Method* et *Parameter*. Un paramètre est décrit par son nom (attribut *name*) et son type (attribut *type*). Lors de l'appel de méthode il est nécessaire de différencier les paramètres d'entrée des paramètres de sortie (respectivement instances des classes *InParameter* et *OutParameter* spécialisant la classe *Parameter*).

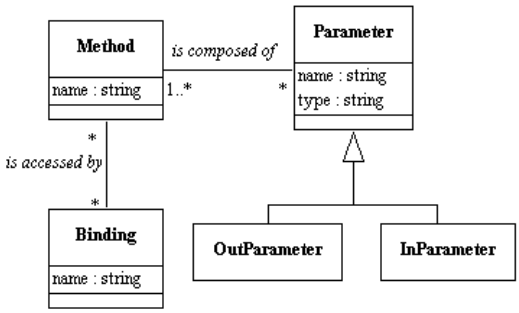


Figure 7.4 Classes et associations du package *Functional Profile*.

Le package *Functional Profile* est utile aux fournisseurs de services lors du processus de description des services. Ce package retranscrit, pour les services issus de la technologie des services Web, la description fonctionnelle réalisée en WSDL. Or, dans l'état de l'art, nous avons souligné que l'utilisation de ce seul type de description ne suffit pas à faciliter les étapes de recherche et de sélection. Dans ce sens, nous proposons le package de description d'informations non fonctionnelles.

7.2.5 Représentation du profil non fonctionnel

La package *Non Functional Profile* décrit les informations qui ne sont pas directement liées à l'appel du service. Ces informations seront exploitées lors des processus de recherche et de sélection et sont déclinées en informations concernant le fournisseur et le fonctionnement du service (le profil de déploiement et les contraintes d'exécution). Ce package est composé de trois classes (*cf.* Figure 7.5), chacune associée à la classe *ConcreteService* du package *Service* : *Provider*, *DeploymentProfile*, *ExecutionConstraint*.

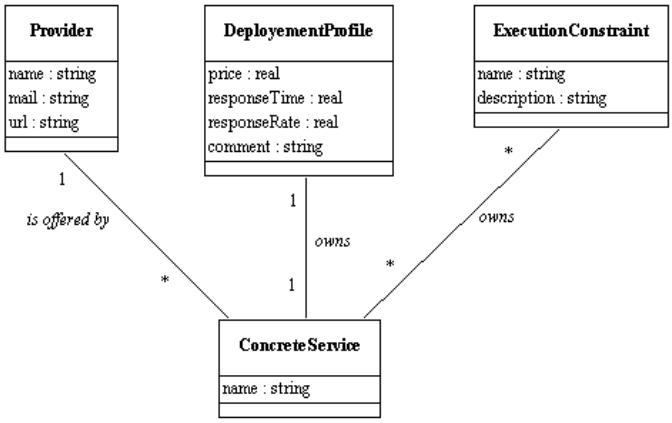


Figure 7.5 Classes et associations du package *Non Functional Profile*.

La classe *Provider*. Cette classe décrit le fournisseur du service. Cette description doit être publiée pour deux raisons. Tout d'abord, le concepteur de système peut avoir besoin de prendre contact avec le fournisseur du service choisi (par exemple, lors d'un problème de connexion). Ensuite, ces informations peuvent aider le concepteur lors du processus de sélection (par exemple si une organisation à l'habitude de travailler avec un fournisseur en particulier). Dans notre modèle, un fournisseur est représenté par son nom, son mail, l'URL représentant, s'il existe, le site du fournisseur.

La classe *DeploymentProfile*. Cette classe comporte les informations concernant le service mais non directement liées à son appel. Ces informations (prix du service, temps de réponse, taux de réponse) doivent être publiées afin de faciliter les étapes de recherche et de sélection réalisées par les concepteurs.

- **Le prix d'un service Web** (attribut *price*) est le montant que l'utilisateur doit payer au fournisseur afin d'avoir le droit de faire appel à ce service. Le prix doit être positif ou nul. La valeur de cet attribut est renseignée par le fournisseur.
- **Le temps de réponse** (attribut *responseTime*) est le temps moyen que met le service Web à envoyer la réponse au client. La valeur de cet attribut peut être, soit renseignée par le fournisseur du service, soit donnée une fois que le service Web a été invoqué une première fois. Il est important de noter que cet attribut n'est donné qu'à titre indicatif mais peut aider le client lors de l'étape de sélection de services.
- **Le taux de réponse d'un service Web** (attribut *responseRate*) est le quotient entre le nombre de réponses donnés par le service Web et le nombre d'appels à ce même service. Ce taux permet d'évaluer la confiance qu'un client peut porter à un service Web. À l'instar du temps de réponse, l'attribut représentant le taux de réponse d'un service Web est renseigné par le fournisseur et n'est donné qu'à titre indicatif.

La classe *ExecutionConstraint*. Cette classe permet de décrire les contraintes d'exécution des services. Chaque contrainte d'exécution est décrite par un couple d'attributs nom et description. Les instances de cette classe facilitent la tâche des clients lors du processus de sélection ou durant l'appel du service. Ce type d'information peut, par exemple, permettre au client de savoir si un service renvoie un résultat seulement si l'utilisateur s'est inscrit préalablement auprès du site du fournisseur. Dans ce cas, l'attribut *name* a pour valeur *registrationRequired* et l'attribut *description* a pour valeur l'URL du site sur lequel le client doit s'inscrire.

Le package *Non Functional Profile* permet d'enrichir la représentation du service lors de l'étape de publication réalisée par les fournisseurs de services.

7.2.6 Représentation du contexte d'utilisation

Le package *Use Context* regroupe les classes (*cf.* Figure 7.7) qui représentent l'information à publier afin de potentiellement mettre en œuvre différents processus d'adaptation (lors de la sélection, de la composition ou de l'exécution) relatifs aux services.

[Kirsch Pinheiro, 2006] a proposé une représentation du contexte d'utilisation qui repose sur deux classes (*cf.* Figure 7.6) : celle représentant la description du contexte (*Description de Contexte*) et celle représentant les différents éléments du contexte (*Elément de Contexte*). La description du contexte d'utilisation est composée d'un ensemble d'éléments du contexte (relation de composition entre les classes *Description de Contexte* et *Elément de Contexte*). L'association (*décrit contexte de*) entre ces deux classes représente le fait que, d'après [Kirsch Pinheiro, 2006], un contexte existe parce qu'il est associé à un élément de contexte particulier (tel que l'utilisateur, le dispositif, etc.). Dans notre travail,

les classes *Description de Contexte* et *Elément de Contexte* sont nommées respectivement *ContextDescription* et *ContextElement*.

Figure 7.6 Description du contexte vu comme une composition d'éléments de contexte, d'après [Kirsch Pinheiro, 2006].

Le travail de [Kirsch Pinheiro, 2006] s'ancre dans le domaine des systèmes collaboratifs. Par conséquent, les classes héritant de la classe *Elément de Contexte* sont spécifiques à ce domaine (telles que les membres, les rôles, les groupes). Dans le modèle de représentation de services Web que nous proposons, le contexte d'utilisation comporte quatre catégories principales (l'utilisateur, le temps, les entités informatiques et la localisation), chacune représentée dans ce package par une classe (respectivement les classes *User*, *Time*, *ComputingEntity*, *Localization*) qui spécialise la classe *ContextElement*. Étant donné que notre travail traite des aspects liés à l'informatique ubiquitaire, nous nous intéressons à l'adaptation des applications à base de services Web aux utilisateurs nomades. De ce fait, nous prenons en compte les spécificités de leur dispositif et de leur environnement en ajoutant la représentation des dispositifs mobiles (classe *MobileDevice* spécialisant la classe *ComputingEntity*).

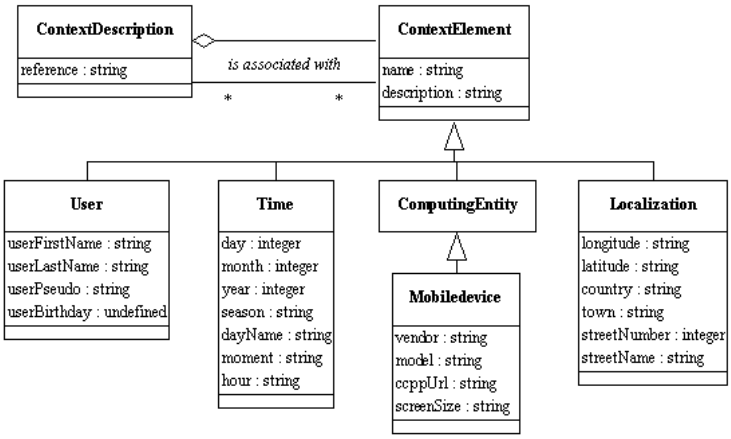


Figure 7.7 Classes et associations du package *Use Context*.

Chacune des classes représentant des éléments du contexte possède un ensemble d'attributs. Ces derniers sont présents ici en tant qu'illustration des classes. Ces attributs peuvent être utilisés ou non par les fournisseurs qui adoptent le modèle de représentation de services Web. Ils sont issus des études faites au préalable sur les travaux portant sur une définition du contexte d'utilisation, tels que [Brown *et al.*, 1997], [Pascoe, 1998], [Chen *et al.*, 2000] et [Dey, 2001].

La classe *User*. L'adaptation de l'information selon les caractéristiques de l'utilisateur est l'une des principales préoccupations de la communauté des hypermédia adaptatifs [Brusilovsky, 2001]. En ce qui concerne le modèle de l'utilisateur, nous ne pouvons ici être exhaustif puisque nous proposons un modèle devant convenir à n'importe quels domaines d'application. Nous considérons seulement l'utilisateur comme un individu (avec son nom, son prénom, un pseudo et sa date de naissance). Or, il est possible d'étendre la classe *User* afin de prendre en compte d'autres éléments. Par exemple, nous pouvons nous baser sur les travaux de [Kirsch Pinheiro, 2006] dans le cadre d'un développement d'applications de travail collaboratif, ou de [Carrillo Ramos, 2007] pour prendre en compte les préférences de l'utilisateur.

La classe *Time*. Nous nous sommes basés sur les travaux de [Pan, 2005] afin d'établir les attributs de la classe *Time* (le jour, le mois, l'année, représentés par un entier ; une saison, le nom du jour, le moment de la journée et une heure, représentés par une chaîne de caractères). Ces travaux [Pan, 2005] modélisent une ontologie du temps et des intervalles de temps, et proposent une extension au langage OWL (OWL-TIME [Hobbs *et al.,* 2006]).

Les classes *ComputingEntity* et *MobileDevice*. Notre travail s'intéressant particulièrement au domaine de l'informatique ubiquitaire, il est important de formaliser le dispositif utilisé qu'il soit mobile ou non. Afin de décrire les capacités des dispositifs mobiles, nous utilisons le langage CC/PP (*Composite Capabilities/Preference Profiles*) [Klyne *et al.,* 2004a], proposé par le W3C. Ce langage est utilisé par l'*Open Mobile Alliance*[65] pour les spécifications des profils des dispositifs mobiles (UAProf – [OMA, 2006]). Aujourd'hui, la plupart des téléphones mobiles possèdent une telle spécification, hébergée sur le Web, que nous utilisons comme attribut de la classe *MobileDevice* (attribut *ccppUrl*). Nous retenons trois attributs du profil (les attributs représentant le fabricant – *vendor*, le modèle de téléphone – *model*, et la taille de l'écran – *screenSize*) étant donné que nous les considérons comme les plus importants. Pour des besoins spécifiques, la description complète du dispositif est accessible via sa description CC/PP.

La classe *Localization*. Le langage GML (*Geography Markup Language*) [OGC, 2007] de l'OGC[66] (*Open Geospatial Consortium*) propose une modélisation de la localisation géodésique (la latitude – représentée par l'attribut *latitude*, et la longitude – représentée par l'attribut *longitude*), la localisation physique (le pays – représentée par l'attribut *country*, la ville – représentée par l'attribut *town*, la rue – représentée par l'attribut *street*, etc.). Nous avons choisi d'utiliser ces caractéristiques afin de déterminer les attributs de la classe *Localization*.

Le package *Use Context* permet d'enrichir la représentation des services lors de l'étape de publication réalisée par les fournisseurs de services, afin de décrire pour quel contexte le service publié est adapté.

Les cinq packages (*Service*, *Application Domain*, *Functional Profile*, *Non Functional Profile* et *Use Context*) composant le modèle de représentation de services Web viennent d'être décrits. La description qui suit définit les relations entre les packages.

7.2.7 Relations entre les packages

Chacun des cinq packages précédemment décrits peut être utilisé indépendamment pour répondre à des besoins spécifiques. Dans WSR-Model il existe un ensemble de liens entre les classes des packages (*cf.* Figure 7.8). Nous étudions, dans un premier temps, les relations impliquant le package *Service*, centre du modèle. Puis nous mettons en évidence les spécificités d'utilisation du package *Use Context* en décrivant les relations qu'il a avec les autres packages.

Relations entre les packages *Service* et *Application Domain*. L'association reliant les classes *ServiceCategory* et *Domain* représente à quel domaine d'application appartient une catégorie de services. Par exemple, la catégorie de services *Localisation* dépend du domaine de la *géomatique*. L'association entre les classes *ConcreteService* et *Functionality* permet aux concepteurs de connaître les services concrets effectuant une tâche spécifique.

Relations entre les packages *Service* et *Functional Profile*. Si le service décrit est un service classique, la seule relation entre ces deux packages associe les classes *ConcreteService* et *Method*. Cette relation permet de représenter les méthodes fournies par le service. Si le service

[65] http://www.openmobilealliance.org/

[66] http://www.opengeospatial.org/

décrit est un service Web, les informations concernant l'accès (classe *Binding*), les méthodes fournies (classe *Method*), les paramètres d'entrée et de sortie (classes *InParameter* et *OutParameter*) afin d'instancier les classes correspondantes sont extraites de la description WSDL. Par conséquent, chacune de ces classes est reliée à la classe *WebService*.

Relations entre les packages *Service* et *Non Functional Profile*. Toutes les classes du package *Non Functional Profile* sont liées à la classe *ConcreteService* puisqu'elles ajoutent un niveau descriptif au service fourni.

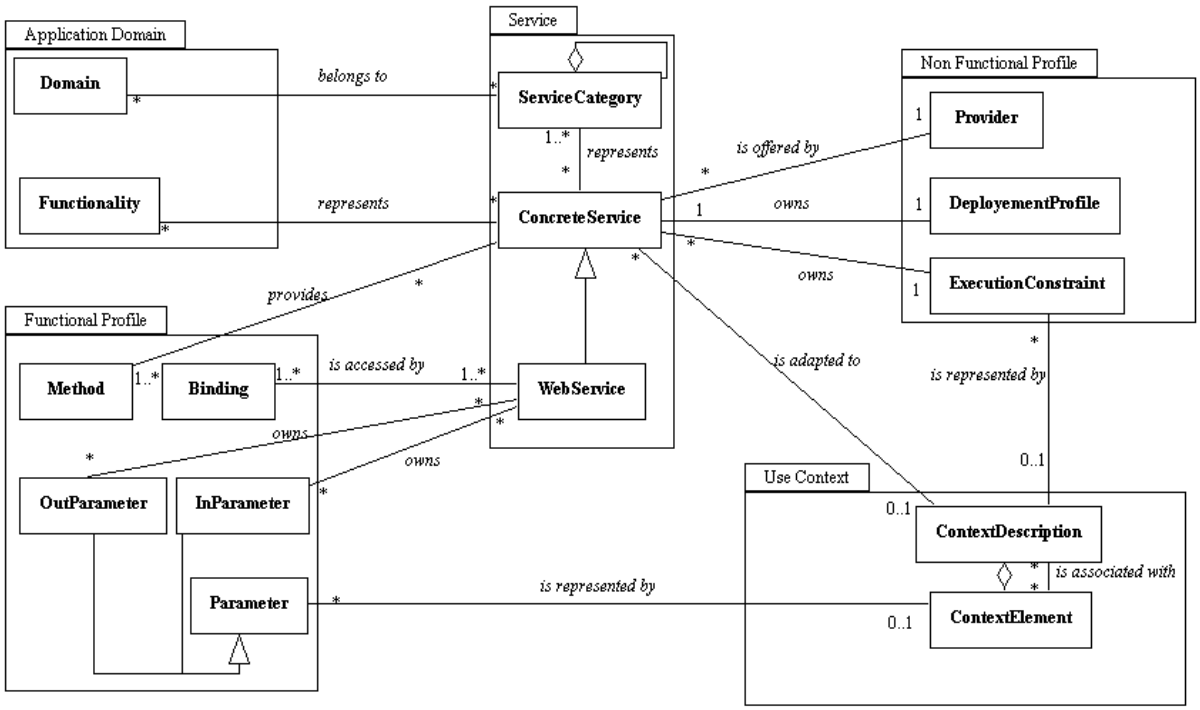


Figure 7.8 Diagramme de classes représentant les relations entre les cinq packages composant le modèle de représentation de services (NB : toutes les classes et associations du modèle ne sont pas représentées sur cette figure).

Relation entre les packages *Use Context* et *Service*. La représentation du contexte d'utilisation permet de caractériser le contexte pour lequel le service décrit est adapté (relation entre les classes *ConcreteService* et *ContextDescription*). Par exemple, un service donné (instance de la classe *ConcreteService*) est développé pour retourner un résultat lisible sur un dispositif mobile de type Sony EricssonK750i (instance de la classe *MobileDevice*).

Relation entre les packages *Use Context* et *Non Functional Profile*. Le contexte d'utilisation permet d'apporter un niveau descriptif supplémentaire aux contraintes d'exécution (relation entre les classes *ExecutionConstraint* et *ContextDescription*). Par exemple, un service retourne au dispositif de l'utilisateur une image lisible sur un écran dont la dimension est d'au moins 176x220. Ceci est formalisé dans le modèle par une instance de la classe *ExecutionConstraint* associé à une instance de la classe *MobileDevice* décrivant la taille de l'écran.

Relation entre les packages *Use Context* et *Functional Profile*. Les paramètres des méthodes décrits à l'aide des classes *Parameter*, *InParameter* et *OutParameter*, du package *Functional Profile*, ne contiennent que les informations concernant leur nom, leur type (attributs de la classe *Parameter*) et le fait qu'ils soient d'entrée ou de sortie (respectivement s'ils sont une instance de la classe *InParameter* ou de la classe *OutParameter*). À l'aide de la relation entre les classes *Parameter* et *ContextElement* un niveau de description sémantique est ajouté aux paramètres.

Considérons un service qui possède deux paramètres d'entrée nommés X et Y, tous les deux de type *string*. Parallèlement, considérons deux instances de la classe *Localization* (sous-classe de la classe *ContextElement – cf.* section 7.2.6) : la première, LT, instancie l'attribut *latitude* ; la seconde, LG, instancie l'attribut *longitude*. Si les instances X et Y sont associées respectivement aux instances LG et LT, alors ceci permettra, par la suite lors de l'appel au service, aux clients de connaître ce que représente chacun des paramètres d'entrée.

L'ensemble des packages décrits et les associations qui les lient forment le modèle de représentation de services Web, WSR-Model, du point du vue fournisseur. Dans la section qui suit, nous proposons de mettre en évidence les changements induits par le point de vue client de ce modèle.

7.3 WSR-Model du point de vue du client

Le modèle de représentation de services Web, WSR-Model, permet de formaliser la requête du client. Cette requête permet de rechercher les services dont le client a besoin afin d'implémenter une application orientée services. La particularité de notre travail est qu'il permet aussi de spécifier la requête en vue de mettre en œuvre des applications adaptées au contexte d'utilisation.

La spécification de la requête du client via WSR-Model a pour conséquence l'ajout d'un nouveau package, nommé *Query Facilities*, décrit tout d'abord, ainsi que des liens entre ce package et ceux représentant le service (*Service*, *Application Domain*, *Functional Profile*, *Non Functional Profile*, et *Use Context*), décrits ensuite.

7.3.1 Représentation de la requête du client

Le package *Query Facilities* de WSR-Model permet de décrire la requête du client ainsi que le client lui-même. Ces aspects sont représentés respectivement par les classes *Query* et *Client* (*cf.* Figure 7.9).

Figure 7.9 Classes et association du package *Query Facilities*.

La classe *Query*. Les instances de cette classe permettent de décrire une requête de services réalisée par des clients. Une requête est représentée par son nom (attribut *name*), la date à laquelle elle a été effectuée (attribut *date*), et si elle a été résolue ou non (attribut *resolved* de type booléen).

La classe *Client*. La classe *Client* représente les clients à l'aide de son nom et de l'adresse de son site Internet (attribut *url*). La relation entre les classes *Client* et *Query* (association *established_by*) permet de connaître les requêtes réalisées par les clients afin de rechercher des services.

Le package *Query Facilities* ne permet pas à lui seul d'exprimer tous les besoins du client en termes de recherche de services. Ce qui suit met en évidence les liens entre ce package et l'ensemble

des autres packages représentant le service. Ainsi ces nouvelles associations enrichissent la représentation de la requête du client.

7.3.2 Relations entre les packages

Afin d'apporter une large expressivité à la représentation de la requête du client, nous avons choisi de lier la classe *Query* du package *Query Facilities* à l'ensemble des autres packages décrivant un service (*Service*, *Application Domain*, *Functional Profile*, *Non Functional Profile*, et *Use Context*). Ce qui suit décrit les classes de chaque package avec lesquelles il est possible de formaliser la requête du client (*cf.* Figure 7.10).

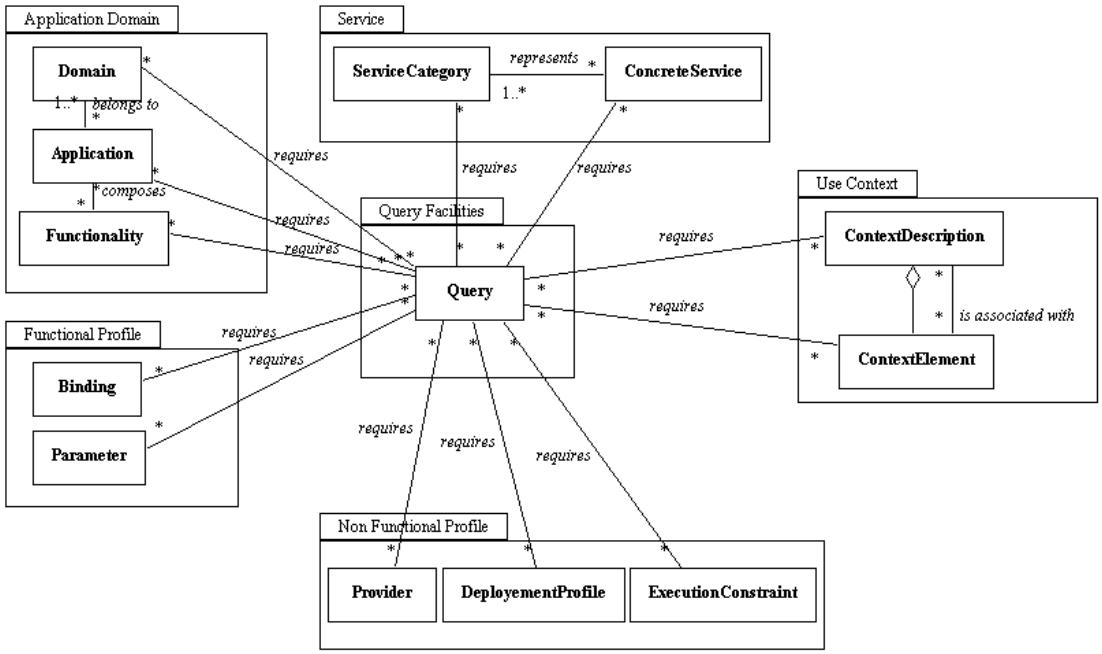


Figure 7.10 Diagramme de classes représentant les relations entre le package *Query* et les packages représentant le service (N.B. : seules les classes et associations mises en jeu pour formaliser la requête sont représentées sur cette figure).

Relations entre le package *Service* et la classe *Query*. La classe *Query* est en relation avec deux classes du package *Service* : la classe *ServiceCategory* et la classe *ConcreteService*. L'association avec la classe *ServiceCategory* permet aux clients de rechercher des services selon leur(s) catégorie(s), tandis que l'association avec la classe *ConcreteService* permet aux clients de rechercher des services concrets. Dans le second cas, il est implicite que les clients connaissent au préalable les noms des services recherchés.

Relations entre le package *Application Domain* et la classe *Query*. La classe *Query* est en relation avec les trois classes du package *Application Domain*. Ce package fournit aux clients un moyen d'exprimer leur requête de service en termes :

- de domaine d'application (via l'association entre la classe *Query* et la classe *Domain*), auquel ils souhaitent que le (ou les) service(s) appartienne(nt).
- de type d'application (via l'association entre la classe *Query* et la classe *Application*), auquel ils souhaitent que le (ou les) service(s) appartienne(nt).

- de fonctionnalités (via l'association entre la classe *Query* et la classe *Functionality*) auxquels ils souhaitent que le (ou les) service(s) réponde(nt).

Relations entre le package Functional Profile et la classe Query. La classe *Query* est en relation avec deux classes du packages *Functional Profile* : les classes *Binding* et *Parameter*. L'association avec la classe *Binding* permet aux clients d'établir un critère de la requête selon le moyen d'accès au service (*i.e.* le protocole à utiliser, tel que SOAP ou HTTP). L'association avec la classe *Parameter* permet aux clients d'ajouter à la requête un critère selon le type des paramètres. La classe *Parameter* étant la classe mère des classes *InParameter* et *OutParameter* (*cf.* section 7.2.4), ce critère de la requête concerne à la fois les paramètres d'entrée et de sortie.

Relations entre le package *Non Functional Profile* et la classe *Query*. La classe *Query* est associée à toutes les classes du package *Non Functional Profile*. Ainsi, le client peut ajouter à la requête de services des critères concernant le fournisseur (via l'association entre la classe *Query* et la classe *Provider*), le profil de déploiement du service (via l'association entre la classe *Query* et la classe *DeploymentProfile*), et les contraintes d'exécution du service (via l'association entre la classe *Query* et la classe *ExecutionConstraint*).

Relations entre le package *Use Context* et la classe *Query*. La classe *Query* est en relation avec les classes *ContextDescription* et la classe *ContextElement*. De ce fait, le client peut enrichir sa requête afin que cette dernière trouve des services adaptés, soit à un contexte donné (via l'association entre la classe *Query* et la classe *ContextDescription*), soit à un ou plusieurs éléments du contexte (via l'association entre la classe *Query* et la classe *ContextElement*).

7.4 Vers l'adaptation au contexte d'utilisation

Un des objectifs de ce travail est de mettre en œuvre de manière transversale l'adaptation au contexte d'utilisation lors de la génération d'applications à base de services Web. Le modèle de représentation de services Web supporte l'adaptation à un contexte d'utilisation décrit de manière explicite par le biais du package *UseContext*. Ceci implique quatre aspects : la représentation enrichie des services Web, la représentation enrichie des requêtes du client, la possibilité de mettre en œuvre aisément des approches et des techniques d'adaptation, ainsi qu'une technique d'adaptation spécifique – la compensation.

Représentation enrichie des services. En utilisant le modèle de représentation de services Web, les fournisseurs de services ont la possibilité de décrire le contexte d'utilisation pour lequel leurs services Web sont adaptés. Cet enrichissement de la représentation des services Web est possible par le biais du package *UseContext*.

Représentation enrichie des requêtes du client. Le package *Query Facilities*, ainsi que les associations qu'il entretient avec les autres packages du modèle WSR-Model, permettent d'enrichir la représentation de la requête du client en termes d'aspects fonctionnels, non fonctionnels, de domaine d'application et de contexte d'utilisation.

Possibilité de mettre en œuvre des approches et des techniques d'adaptation. L'utilisation de WSR-Model lors de l'étape de description facilite l'étape de sélection via l'enrichissement de la représentation des services. Ainsi WSR-Model permet de mettre en œuvre la **technique d'adaptation par sélection de services**. Cette technique d'adaptation permet de réaliser deux formes d'adaptation : l'**adaptation de la demande de services** et l'**adaptation de la composition** (*cf.* section 6.4.2).

Possibilité de mettre en œuvre la technique d'adaptation par compensation. Nous définissons une technique d'adaptation non utilisée par les travaux étudiés dans le chapitre

concernant l'adaptation et les services Web (chapitre 5) qui est la technique d'adaptation par compensation. Cette technique d'adaptation consiste, lors de l'échec de l'invocation d'un service, à invoquer un service équivalent. Des services sont équivalents s'ils répondent aux mêmes attentes du client (par exemple, même type de paramètre d'entrée, même fonctionnalité, etc.). Par l'intermédiaire du modèle de représentation de services Web il est aisé de regrouper les services Web selon l'ensemble des critères présents dans le modèle (telles que la catégorie du service avec le package *Service* ou la fonctionnalité du service avec le package *ApplicationDomain*). De ce fait, nous pouvons regrouper des services et ainsi connaître les services équivalents selon des attributs demandés par le client. Ceci permet de mettre en œuvre la technique d'adaptation par compensation.

De manière plus détaillée, nous remarquons que WSR-Model, via les packages *Service* et *Use Context* (*cf.* Figure 7.11), permet de décrire trois types de services (les *services classiques*, les *services adaptés*, et les *services d'adaptation*). Les deux derniers types de services (les *services adaptés* et les *services d'adaptation*) sont des acteurs de la mise en œuvre de l'adaptation au contexte d'utilisation.

Les services classiques. Les services classiques sont des services qui répondent à une fonctionnalité spécifique et n'apportent pas d'adaptation. Ce type de service ne possède pas d'association entre les classes *ConcreteService* et *ContextDescription*.

Les services adaptés. Un service adapté est un service dont la fonctionnalité principale n'est pas d'adapter, mais dont le résultat est adapté par rapport à un (ou plusieurs) élément(s) du contexte d'utilisation. Dans ce cas, il existe une instance de la classe *ContextDescription*, représentant le contexte d'utilisation auquel le service est adapté, à laquelle l'instance de la classe *ConcreteService*, représentant le service, est associée.

Les services d'adaptation. Un service d'adaptation est un service conçu spécifiquement pour mettre en œuvre un type d'adaptation à un contexte d'utilisation donné (par exemple en adaptant les résultats d'un premier service pour qu'il soit en adéquation avec ce contexte d'utilisation). Dans ce cas, l'instance de la classe *CategoryService*, à laquelle l'instance de ce service est liée, est obligatoirement *Adaptation*. De plus, l'instance de ce service est associée à l'instance de la classe *ContextDescription* à laquelle il est adapté.

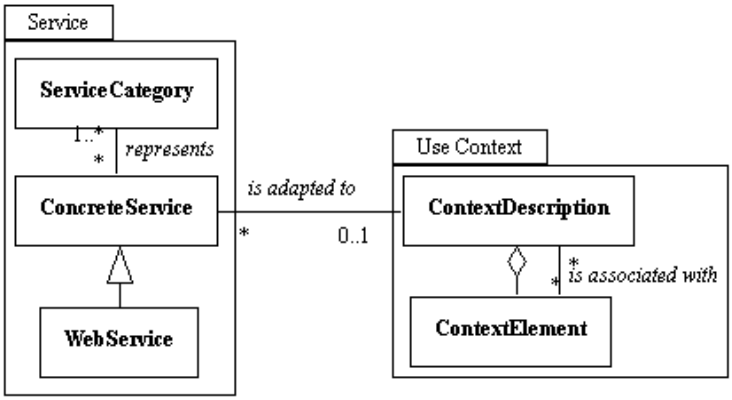


Figure 7.11 Diagramme de classes rappelant les relations entre les packages *Service* et *Use Context*.

Ces types de services peuvent être implémentés par la technologie des services Web (*cf.* la classe *WebService*).

7.5 Conclusion

Afin que les services Web pré-établis puissent être visibles et ainsi être utilisés (et réutilisés), les fournisseurs doivent les décrire (à l'aide de WSDL s'il s'agit d'un service Web) et les publier dans un registre. Ainsi, il existe des registres dans lesquels sont enregistrées les représentations de services mis à la disposition par leur fournisseur. Dans la plupart des registres existants, les représentations des services ne prennent en compte que la localisation du service et le site Web du fournisseur. Cette représentation est, de notre point de vu, insuffisante afin de réutiliser, rechercher, et sélectionner les services Web.

Afin de faciliter la mise à disposition des services Web par les fournisseurs et de faciliter leur utilisation par les clients nous proposons un modèle enrichi de représentation de services, WSR-Model. Ce dernier contient des informations à propos du **service** lui-même, du **fournisseur**, du **domaine d'application** dans lequel le service peut être utilisé et le **contexte d'utilisation** auquel le service peut répondre de manière adaptée. Les quatre entités à prendre en compte (service, fournisseur, domaine d'application, et contexte d'utilisation) sont représentées dans WSR-Model par un ensemble de classes (regroupées dans des packages). Les informations concernant le service sont représentées par deux packages : le **package *Service***, qui prend en compte les données élémentaires du service et le **package de profil fonctionnel (*Functional Profile*)** dans lequel le client trouve des informations concernant l'utilisation du service (méthodes à appeler, protocoles à utiliser, etc.). Le **package de profil non fonctionnel (*Non Functional Profile*)** décrit les données relatives au fournisseur du service à l'aide de la description du fournisseur lui-même, des contraintes d'exécution, et du profil de déploiement du service. Enfin, le contexte d'utilisation auquel peut répondre le service décrit est représenté par le **package *Use Context***.

WSR-Model propose aussi une vue client afin de permettre à ce type d'acteur d'enrichir l'expressivité de leurs requêtes de services. Le client peut inclure dans sa requête une demande en termes :

- du domaine d'application auquel le client souhaite que les services Web trouvés appartiennent (avec le package *Application Domain*) ;
- de la catégorie à laquelle le service recherché doit appartenir (avec le package *Service*) ;
- des propriétés fonctionnelles (avec le package *Functional Profile*) et non fonctionnelles (avec le package *Non Functional Profile*) que le service doit posséder ;
- du contexte d'utilisation auquel le service doit être adapté (avec le package *Use Context*).

L'extensibilité de WSR-Model repose sur sa formalisation à l'aide d'une représentation objets. Ainsi, si lors de son utilisation des fournisseurs ou des clients désirent ajouter des classes ou des attributs (afin de, par exemple, étendre le package *Use Context*), ceci est aisé.

Le modèle de représentation de services Web permet par son utilisation l'**adaptation au contexte d'utilisation** à travers cinq points :

- Ce modèle permet l'**enrichissement des services Web** en y incluant le contexte d'utilisation pour lequel les services sont adaptés ;
- Ce modèle permet **l'enrichissement de la requête du client** à l'aide de l'ensemble des packages de WSR-Model ;
- Ce modèle offre la possibilité de **mettre en œuvre des approches d'adaptation** (telles que l'adaptation de la demande de services et l'adaptation de la composition) **et des technique d'adaptation** (telle que la sélection de services Web) par le biais de l'enrichissement de la représentation des services Web ;

- Ce modèle offre la possibilité de **mettre en œuvre une technique d'adaptation** que nous appelons **compensation**. Ceci est permis du fait que WSR-Model fournit un moyen d'établir des groupes de services équivalents selon différents critères (tels que la catégorie, la fonctionnalité fournie).
- Ce modèle offre le moyen de décrire deux acteurs de la mise en œuvre de l'adaptation au contexte d'utilisation : les *services adaptés* et les *services d'adaptation*.

Grâce à l'enrichissement de la représentation d'un service Web proposé par WSR-Model, nous facilitons les processus de recherche et de sélection de services dans le cadre de l'implémentation d'applications adaptées au contexte d'utilisation. Cependant, les applications à implémenter ne reposent pas sur un unique service Web mais bien souvent sur la composition d'un ensemble de services. Or, d'après l'étude faite dans l'état de l'art, aucune solution de composition de services Web ne couvre toutes nos attentes (intégration de la sémantique et de l'adaptation au contexte d'utilisation, évolutivité de la composition, et compensation de services). C'est pourquoi nous proposons un modèle de composition de services Web présenté dans le chapitre suivant.

8 MODÈLE DE COMPOSITION DE SERVICES WEB PAR MÉTHODE DE RÉSOLUTION DE PROBLÈMES

L'étude de l'état de l'art présenté dans la première partie de ce mémoire met en relief le fait que les travaux portant sur la composition de services Web présentent des lacunes au niveau de la définition de la composition de services (activité *Assemblage de services*), de l'évolutivité de la composition, de la surveillance (activité *Contrôle de l'exécution*) et de la compensation de services (qui englobe les activités *Évolutivité de la composition* et *Recherche de services*). Le premier point (définition de la composition de services) est régi par les modèles de composition tandis que les trois derniers (évolutivité de la composition, surveillance de la composition et compensation de services) sont mis en œuvre par les plates-formes de composition. Dans ce chapitre, nous répondons au premier point et décrivons le modèle de composition de services Web *ProbCWS* (*Problem-solving for Composition of Web Services*). Les trois points suivants sont traités dans le chapitre portant sur la plate-forme de génération d'applications à base de services Web (chapitre 11).

Dans les chapitres portant sur la composition de services Web et sur la synthèse de l'état de l'art, nous avons mis en évidence le fait que les langages de définition de la composition doivent reposer sur des outils formels afin de bénéficier des mécanismes de raisonnement sous-jacents à ces derniers. Notre modèle s'appuie sur un outil de résolution de problèmes issu du domaine de l'intelligence artificielle, et, plus particulièrement, sur la résolution de problèmes par modèle de tâches [Chandrasekaran, 1990].

Dans ce chapitre, nous donnons, dans un premier temps, notre propre définition du cycle de vie d'une composition de services Web qui nous permet d'identifier par la suite les besoins à prendre en compte dans notre modèle. Ensuite, nous montrons comment l'approche de la résolution de problèmes par modèle de tâches permet de répondre aux besoins précédemment définis. Par la suite, nous définissons la structure et les concepts de ProbCWS qui permet de mettre en parallèle le concept de résolution de problèmes et la composition de services Web. Nous avons choisi de réaliser un arbre de résolution spécifique à ProbCWS afin, entre autres, de définir graphiquement la définition de la composition et les flux de données. Cet arbre de résolution est étudié avant de décrire les apports de ProbCWS en termes d'adaptation au contexte d'utilisation.

8.1 Définition du cycle de vie d'une composition de services Web

D'après [Benatallah *et al.,* 2002], le cycle de vie d'une composition de services Web est basé sur l'analyse et la conception de sept activités (*cf.* section 4.1.1 – chapitre 4). Les trois premières activités (*Encapsulation de services natifs*, *Publication de service* et *Établissement d'accord d'externalisation*) sont de notre point de vue incluses dans le cycle de vie d'un service élémentaire. Les quatre dernières activités (*Assemblage des services composants*, *Exécution des services composants*, *Contrôle de l'exécution*, *Évolutivité de la composition*) appartiennent à notre définition d'un cycle de vie d'une composition de services Web. Nous ajoutons à ces quatre activités une cinquième nommée *Recherche de services*, entre les activités *Assemblage des services composants* et *Exécution des services composants* (*cf.* Figure 8.1). L'ensemble des activités du cycle de vie d'une composition de services Web est illustré par un diagramme d'activités (*cf.* Figure 8.1). Les différentes activités et les transitions sont décrites dans ce qui suit.

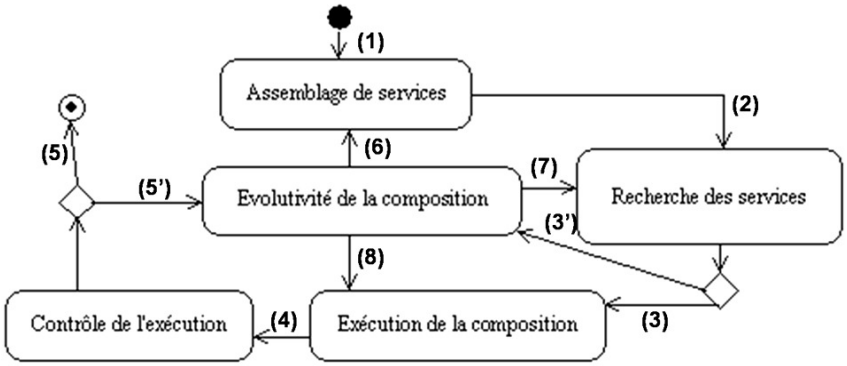


Figure 8.1 Diagramme d'activités UML représentant le cycle de vie d'une composition de services Web, inspiré de [Benatallah *et al.*, 2002].

Assemblage des services composants. Cette activité permet de spécifier l'ensemble des catégories de services à composer afin d'atteindre l'objectif suivi dans la composition de services Web. Cet assemblage comporte une phase d'identification des catégories de services et de spécification de leurs interactions. Lors de la définition de l'assemblage des services Web, nous pensons qu'il est pertinent d'utiliser des outils pré-existants (à l'instar de OWL-S [Martin et al., 2004] utilisant la logique de description [Baader et al., 2003] ou de WS-CDL [Kavantzas et al., 2005] utilisant le π-calcul [Milner, 1999]). Cette activité est déclenchée par la requête initiale **(1)** (formulée, soit par un utilisateur, soit par un logiciel) qui est transmise, soit au moteur d'orchestration (dans le cadre d'une orchestration de services Web), soit à un premier service que nous nommons SW (dans le cadre d'une chorégraphie de services Web).

Recherche de services. Cette activité est en charge de trouver un service correspondant à la catégorie de services identifiée au niveau de l'*Assemblage de services* **(2)**. Il s'agit, soit du moteur d'orchestration, soit du service SW, qui recherche les services Web à invoquer.

Exécution des services composants. Cette activité consiste en l'exécution des spécifications de la composition précédemment définies en vérifiant si les contraintes d'exécution entre chaque service Web sont respectées **(3)**. Selon le type de composition, soit le moteur d'orchestration, soit le service SW exécute la première tâche de la composition (définie lors de l'activité *Assemblage de services*) en invoquant le service trouvé dans l'activité précédente (*Recherche de services*).

Contrôle de l'exécution des services composants. Cette activité supervise l'exécution de la composition en vérifiant, par exemple, l'accès aux services, les changements de statut, les

échanges de messages. Ce contrôle permet de détecter des violations de contrats, de mesurer les performances des services appelés, et de permettre l'évolutivité de la composition si des services Web échouent lors de l'exécution **(4)**. Selon le type de composition, cette activité est réalisée, soit par le moteur d'orchestration, soit par le service qui précède, dans le cadre d'une chorégraphie. Si tous les services ont été appelés avec succès et que la composition a été réalisée, soit le moteur de composition (pour une orchestration de services), soit le dernier service appelé (pour une chorégraphie), retourne le résultat de la composition à l'utilisateur (humain ou logiciel) qui a formulé la requête initiale **(5)**.

Évolutivité de la composition. Cette activité permet de faire évoluer la composition en supprimant les défauts de l'organisation de services, en utilisant de nouveaux services, ou en prenant en compte les retours de la phase de contrôle. Si les activités de recherche **(3')** ou de contrôle **(5')** échouent, soit le moteur d'orchestration, soit le service SW, tente de pallier ces échecs en faisant évoluer la composition de services initialement définie (retour à l'activité *Évolutivité de la composition*). Cette activité consiste :

- soit à redéfinir la composition en compensant une tâche qui ne peut être exécuter (retour à l'activité *Assemblage de services* si l'activité *Recherche de services* a échoué) **(6)** ;
- soit à rechercher un nouveau service pour une tâche qui n'a pas pu être résolue (retour à l'activité *Recherche de services* si l'activité *Contrôle de l'exécution* a échoué) **(7)** ;
- soit à exécuter à nouveau la composition si une contrainte n'avait pas été respectée (retour à l'activité *Exécution de la composition* si l'activité *Contrôle de l'exécution* a échoué) **(8)**.

8.2 Identification des besoins en termes de composition de services

Dans cette section, nous identifions les besoins à satisfaire dans le cycle de vie d'une composition de services Web afin de proposer une solution qui répond aux problématiques sous-jacentes de la composition (telles que la compensation ou la surveillance de l'exécution) et met en œuvre l'adaptation au contexte d'utilisation. À la suite de l'étude réalisée dans l'état de l'art sur les langages de composition de services Web, nous pouvons dire que l'activité du cycle de vie portant sur l'*Assemblage de services* doit être améliorée afin de proposer une définition de la composition capable d'aider les concepteurs à mettre en œuvre des applications orientées services Web adaptées au contexte d'utilisation.

L'étape d'*Assemblage de services* est composée d'une phase d'identification des catégories de services à appeler dans la composition, et d'une phase de définition des interactions entre les services composants. Nous pensons que les améliorations de cette étape du cycle de vie portent sur ces deux phases et consistent à *utiliser des outils formels existants*, à *utiliser des approches sous-jacentes au Web sémantique*, et à *intégrer le concept d'adaptation au contexte d'utilisation*.

Utilisation d'un outil formel existant pour définir la composition. Le fait de réutiliser des outils formels existants en les appliquant à la composition de services Web permet de résoudre des problèmes sous-jacents à ce concept, tels que la surveillance de l'exécution ou la mise en œuvre de la compensation de services. Prenons l'exemple de l'outil formel π-calcul [Milner, 1999] qui a l'avantage de redéfinir la structure d'un processus de manière dynamique au fur et à mesure de sa progression. Le langage WS-CDL [Kavantzas *et al.,* 2005] qui se base sur ce concept peut ainsi mettre en œuvre une définition dynamique de la composition de services.

Utiliser des approches sous-jacentes au Web sémantique. Afin d'améliorer le processus de recherche de services, il est préférable de prendre en compte la sémantique du service. Si la définition de la composition ne prend en compte que les types d'entrée et de sortie des services

composants lors de l'identification des services, la composition risque de ne pas convenir au client. Prenons l'exemple d'une requête de composition qui consiste à implémenter un Système d'Information Géographique qui affiche une carte dont le centre est la localisation courante de l'utilisateur. Si l'étape d'*Assemblage de services* ne prend en compte que l'aspect syntaxique lors de l'identification de services, des services ne répondant pas à l'attente du client peuvent intervenir dans la composition. Dans notre exemple, supposons que l'activité d'*Assemblage de services* fasse intervenir dans la composition un service d'*affichage de carte*, qui a pour paramètre d'entrée un couple de réel (représentant les coordonnées GPS) et en sortie un paramètre de type SVG [Ferraiolo *et al.*, 2003] (représentant la carte résultat). Si l'activité d'*Assemblage de services* ne prend pas en compte la sémantique du service, des services intervenant dans la composition peuvent convenir en termes de paramètres d'entrée et de sortie mais ne fourniraient pas la fonctionnalité attendue. Dans notre exemple, ce peut être un service qui selon deux valeurs de type réel renvoie une image vectorielle de type SVG représentant un monument près de la localisation et non une carte. Il est donc préférable d'identifier les services composants en intégrant un niveau de sémantique. Ainsi, dans notre exemple, l'activité d'*Assemblage de services* continue d'identifier des services dont les paramètres d'entrée sont de type réel et le paramètre de sortie de type SVG, mais doit également savoir que les paramètres d'entrée désignent des coordonnées GPS et le paramètre de sortie une carte.

Intégration de l'adaptation au contexte d'utilisation. Une de nos préoccupations est de mettre en œuvre l'adaptation au contexte d'utilisation au sein des applications orientées services Web. Le contexte d'utilisation que nous considérons est celui intégré dans la représentation de services Web que nous avons proposée précédemment (*cf.* section 7.2.6), c'est-à-dire que le contexte d'utilisation est composé de la définition de l'utilisateur, sa localisation, le moment de l'utilisation de l'application, et son dispositif. Afin de mettre en œuvre l'adaptation de manière transversale (dans toutes les étapes du cycle de vie d'une application à base de services Web), le modèle de composition de services Web doit intégrer dans la définition de la composition des éléments dédiés à l'adaptation au contexte d'utilisation.

Afin d'offrir une solution de définition de la composition de services Web, notre proposition doit utiliser des outils formels, intégrer de la sémantique à la définition, et associer le concept d'adaptation au contexte d'utilisation à l'activité d'*Assemblage de services*.

8.3 Outil formel pour définir la composition

L'étude et la synthèse de l'état de l'art ont montré que la puissance d'expression d'un modèle de composition dépend, en grande partie, de l'outil formel utilisé pour définir la composition de services Web. Parmi les approches utilisées par les travaux étudiés nous comptons :

- π-calcul [Milner, 1999] utilisé par le langage WS-CDL [Kavantzas *et al.,* 2005]. Cette approche a l'avantage de redéfinir la structure d'un processus de manière dynamique au fur et à mesure de sa progression et ainsi proposer l'évolutivité de la composition. Cependant, dans cette approche, l'intégration du contexte d'utilisation au sein de la définition de la composition n'est pas considérée.
- Les *workflows* [WMC, 1999] utilisés par le langage BPEL4WS [Andrews *et al.,* 2003] et la plate-forme de composition SELF-SERV [Sheng *et al.,* 2002]. Cette approche ne permet de définir que des compositions rigides, donc non évolutives.
- Les méthodes de résolution de problèmes (telles que le réseaux de tâches hiérarchiques [Erol *et al.,* 1994]) utilisées par les plates-formes de composition SHOP2 [Sirin *et al.,*

2004] et IRS-III [Cabral *et al.,* 2006]. Cette approche propose un moyen de combiner la définition de compositions évolutives et l'intégration du contexte d'utilisation.

La comparaison faite entre les différentes approches utilisées dans les travaux de composition de services Web nous a poussé à choisir la méthode de résolution de problèmes par modèle de tâches. Cette section décrit cette dernière approche. Nous présentons ensuite un argumentaire sur le choix de cette approche pour définir la composition de services Web.

8.3.1 Définition des méthodes de résolution de problèmes par modèle de tâches

[Russell *et al.*, 1995] répertorient quatre courants de systèmes issus du domaine de l'intelligence artificielle :

- Les systèmes qui pensent comme les humains ;
- Les systèmes qui agissent comme les humains ;
- Les systèmes qui pensent de manière rationnelle ;
- Les systèmes qui agissent de manière rationnelle.

Notre proposition, et en général les méthodes de résolution de problèmes, s'ancre dans la première catégorie de systèmes (*les systèmes qui pensent comme les humains*) et ainsi peut se référencer, d'après [Russell *et al.*, 1995], à la définition de l'intelligence artificielle donnée par [Bellman, 1978] : *l'intelligence artificielle est l'automatisation des activités associées à la pensée humaine telles que la prise de décision et la résolution de problèmes*.

Dans la suite de la description des méthodes de résolution de problèmes, nous allons dans un premier temps étudier les méthodes de résolution de problèmes dans leur globalité, puis, les méthodes de résolution de problèmes par modèle de tâches en particulier.

8.3.1.1 Méthode de résolution de problèmes

L'objectif des méthodes de résolution de problèmes est de décrire le processus de raisonnement des systèmes à base de connaissances de manière indépendante de leur implémentation [Gómez Pérez *et al.*, 1999]. Toujours d'après ces auteurs, trois points caractérisent une méthode de résolution de problèmes :

(1) Une méthode de résolution de problèmes **spécifie les étapes d'inférence** afin d'atteindre l'objectif donné.

(2) Une méthode de résolution de problèmes **définit les structures de contrôles** en plus des étapes d'inférence.

(3) **Les rôles joués par les connaissances du domaine** dans chacune des étapes doivent être spécifiés. Il existe deux types de rôles : les rôles statique et dynamique.

- *Le rôle statique* décrit la connaissance du domaine utilisé par la méthode de résolution de problèmes.
- *Le rôle dynamique* formalise les entrée(s) et sortie(s) des étapes d'inférence.

D'après [Willamowski, 1994], résoudre un problème consiste à choisir des actions appropriées parmi un ensemble d'actions possibles, à les organiser dans un plan d'actions, et à les exécuter. Cette résolution de problèmes peut être supportée par deux phases de résolution distinctes : la phase de planification et la phase d'exécution.

La phase de planification. Cette phase consiste à déterminer au préalable l'organisation des actions à exécuter. L'inconvénient de ce processus est que si une action échoue tout le plan échoue.

La phase d'exécution. Lors de la résolution de problèmes, chaque action est choisie une à une jusqu'à obtenir le résultat désiré. Ceci permet, si une action est indisponible, ou si une action échoue de faire des retours en arrière, ou d'essayer des actions alternatives. Cependant, si aucune organisation globale n'est définie au préalable, cette phase d'exécution risque de ne pas être efficace, puisque beaucoup d'actions non adaptées peuvent être exécutées.

La résolution de problèmes est utilisée afin de développer des systèmes dits experts. Un système expert est un système de résolution automatique de problèmes, conçu pour atteindre les performances d'experts humains dans des domaines spécifiques, à partir d'un ensemble de connaissances acquises au préalable auprès d'experts [Haton *et al.,* 1991]. D'après [Horn *et al.,* 1991] ces types de systèmes comportent trois inconvénients :

(1) ces systèmes ne donnent une réponse exacte que dans un **domaine spécifique** ;

(2) le raisonnement qui a permis la résolution du problème n'est pas expliqué de manière satisfaisante (*i.e.* compréhensible) pour l'utilisateur, du fait d'un **formalisme de représentation de connaissances limité** ;

(3) **le processus d'acquisition de connaissances et leur formalisation n'est pas suffisamment étudié et structuré**.

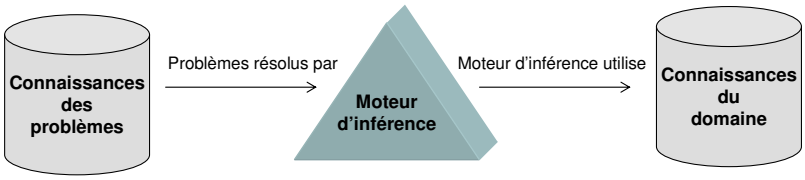


Figure 8.2 Architecture des applications utilisant des méthodes de résolutions de problèmes, d'après [Gómez Pérez *et al.,* 1999].

Afin de pallier ces inconvénients il existe deux approches : la *modélisation fonctionnelle* des connaissances qui identifie un ensemble de problèmes existants (expérimentée dans les travaux de [Chandrasekaran *et al.*, 1988]) et la *modélisation conceptuelle* des connaissances qui identifie les éléments de connaissances à prendre en compte pour la résolution du problème (mise en œuvre dans KADS – *Knowledge Acquisition and Documentation Structuring* [Breuker *et al.*, 1988] [Wielenga *et al.*, 1986]). Ceci convient à l'architecture des méthodes de résolutions de problèmes définis plus tard par [Gómez Pérez *et al.*, 1999] : l'architecture de telles méthodes doit décrire ce que la méthode peut résoudre (les problèmes), comment l'atteindre (les étapes d'inférence) et de quoi elles ont besoin pour l'atteindre (les tâches élémentaires issues des connaissances du domaine). Ainsi les types d'applications utilisant des méthodes de résolutions de problèmes séparent la gestion des connaissances (base de connaissances du domaine), le moteur d'inférence, et les définitions des problèmes (base de connaissances des problèmes) – *cf.* Figure 8.2.

Ce qui suit illustre les spécificités des méthodes de résolution de problèmes par modèle de tâches.

8.3.1.2 Méthode de résolution de problèmes par modèle de tâches

Un des modèles existant afin de résoudre un problème est le *modèle de tâches* [Chandrasekaran, 1990]. Le modèle de tâches consiste à décomposer le problème en tâches, elles-mêmes décomposables en sous-tâches. Les tâches élémentaires sont des méthodes (programmes) simples à exécuter.

La résolution de problèmes par modèle de tâches se rapproche de la *planification hiérarchique* [Sacerdoti, 1974] [Cohen *et al.*, 1982]. Ce type de planification est déjà utilisé dans le domaine de la composition de services Web (*cf.* chapitre 4). Il s'agit de prendre en compte les avantages des deux phases de résolution de problèmes (*phase de planification* et *phase d'exécution – cf.* section précédente) dans un contexte d'utilisation de systèmes à base de connaissances. Cette approche décrit plusieurs niveaux d'abstraction (définit et structure le raisonnement à différents niveaux de précision) et de décomposition (décomposition récursive en (sous-)problèmes de plus en plus élémentaires).

Afin d'étudier les spécificités du modèle de tâches, il est important de définir les termes de tâche, de stratégie et de plan d'actions (d'après [Ferreira da Cunha, 1999]).

La tâche. Une tâche représente un problème à résoudre (selon des entrées et des sorties) et une stratégie (*cf.* ci-dessous). Il existe trois types de tâche : les tâches décomposables, spécialisables, et élémentaires.

- **La tâche décomposable**. Une tâche décomposable représente un problème pour lequel la stratégie de résolution est composée en une *séquence* de tâches.
- **La tâche spécialisable**. Une tâche spécialisable est une tâche de haut-niveau, trop générale. Ce type de tâche permet de regrouper un ensemble de sous-tâches qui répond au même problème (ou ayant des caractéristiques communes) mais ces sous-tâches contiennent des spécificités. Lors de la résolution, la sous-tâche choisie est celle qui correspond le mieux au problème posé.
- **La tâche élémentaire**. Une tâche élémentaire peut être réalisée directement. La stratégie qui lui est associée est l'exécution d'un module connu, correspondant, soit à un programme, soit à une tâche dont la stratégie a déjà été définie.

La stratégie. La stratégie définit la manière dont le problème peut être résolu en représentant la décomposition d'une tâche en sous-tâches (une sous-tâche peut être elle-même décomposable, spécialisable ou élémentaire).

Le plan d'actions. Un plan d'actions est une séquence de tâche(s) élémentaire(s) exécutée(s) lors de la résolution effective du problème.

Le plan d'action est déterminé par la structure de la tâche qui est, soit décomposable, soit spécialisable. D'après [Crampé, 1994], la détermination d'un plan d'actions suit de façon récurrente le schéma suivant :

Si T est une tâche décomposable, alors le plan d'actions est l'enchaînement des sous-plans représentés par les sous-tâches qui composent T.

Si T est une tâche spécialisable, alors il faut choisir une sous-tâche parmi celles qui spécialisent T, celle qui convient le mieux, ainsi le plan d'action est le sous-plan représenté par cette sous-tâche.

Si T est une tâche élémentaire, alors le plan d'actions correspond à la réalisation d'une seule étape.

La résolution du problème peut être représentée par un *arbre de résolution* dont la racine est la tâche principale (le problème), les arcs représentent la structure de la tâche (décomposition ou spécialisation), les nœuds sont, soit des tâches décomposables, soit spécialisables, et les feuilles sont des tâches élémentaires (*cf.* Figure 8.3). Dans l'exemple illustré par la Figure 8.3, le problème (T) est décomposable en trois sous-tâches (T_1 – tâche élémentaire, T_2 – tâche spécialisable et T_3 – tâche décomposable). La tâche T_2 est spécialisée par les sous-tâches élémentaires T_4 et T_5. La tâche T_3 est décomposée en la sous-tâche élémentaire T_6 et la sous-tâche spécialisable T_7. Enfin, la tâche T_7 est

spécialisée en deux sous-tâches élémentaires T_8 et T_9. Le plan d'actions de cette résolution de problème est composé de quatre tâches : T_1, T_4 ou T_5, T_6, T_8 ou T_9.

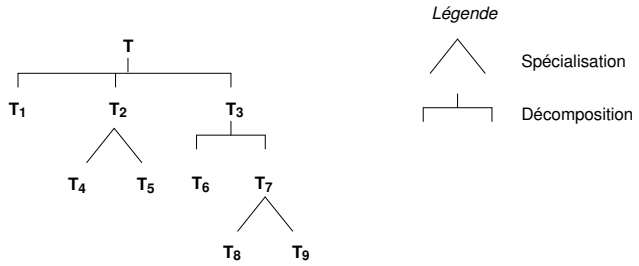


Figure 8.3 Exemple d'un arbre de résolution d'une résolution de problème (issu de la tâche T) par modèle de tâches.

L'architecture sous-jacente à la mise en œuvre de méthode de résolution de problèmes décrite plus haut (*cf.* Figure 8.2) favorise la réutilisation des tâches élémentaires exécutées par des programmes simples en séparant les connaissances de problèmes et les connaissances du domaine (*i.e.* les tâches élémentaires).

Le paragraphe qui suit expose un rapprochement entre les méthodes de résolutions de problèmes par modèle de tâches et la composition de services Web en se basant sur les besoins exposés plus haut (*cf.* section 8.2).

8.3.2 Argumentaire sur le choix de l'approche

Nous venons de présenter le concept de méthodes de résolution de problèmes par modèle de tâches. Nous listons ici les trois points qui ont guidé notre choix pour cet outil formel en vue de définir notre modèle de composition de services Web. Notre argumentaire s'appuie tout d'abord sur le fait que cette méthode nous permet de conceptualiser la requête de la composition de services Web comme un problème à résoudre. Ensuite, les deux points suivants découlent de nos besoins, c'est-à-dire l'intégration de la sémantique et de l'adaptation au contexte d'utilisation au sein de la définition de la composition de services Web.

La composition de services Web vue comme un problème à résoudre. D'après [Wagner *et al.,* 2002], afin de résoudre un problème complexe il est préférable de le décomposer en sous-problèmes. Ces auteurs appliquent ceci à la mise en œuvre de systèmes d'information mobiles. Ainsi, le système (le problème complexe) est implémenté par un ensemble de services élémentaires (sous-problèmes). Le concept de résolution de problèmes par modèle de tâches [Chandrasekaran, 1990], qui repose sur la décomposition récursive du problème principal, peut répondre à ces attentes d'un point de vue théorique. L'utilisation de méthodes de résolution de problèmes par modèle de tâches afin de réaliser le modèle de composition de services Web permet de :

- **Conceptualiser la requête de composition de services comme un problème**, décomposé en tâches, chacune des tâches élémentaires étant exécutée par l'appel d'un service Web.
- **Conceptualiser la composition de services Web comme le plan d'actions de la résolution du problème.**

L'intégration de la sémantique dans le processus de composition de services. Le modèle de résolution de problèmes par modèle de tâches repose sur la séparation entre les connaissances du domaine sur lesquelles on va pouvoir raisonner afin d'appliquer la méthode de résolution et les

tâches à exécuter. Ainsi, nous pouvons intégrer un niveau de sémantique dans ce type de modèle étant donné que les outils existants en gestion de connaissances permettent de stocker et raisonner à ce niveau. De plus, les tâches concrètes (dans notre cas, les services Web élémentaires) sont indépendantes du raisonnement et peuvent être réutilisées. L'utilisation de méthodes de résolution de problèmes par modèle de tâches afin de réaliser notre proposition permet de :

- **Séparer de manière claire la base théorique** (connaissances du domaine et raisonneur) et **les composants exécutables et réutilisables** (les tâches élémentaires en général, et les services Web dans notre cas) [Motta *et al.*, 1999].

L'intégration du concept d'adaptation au contexte d'utilisation dans la composition. Nous avons vu dans le chapitre portant sur l'adaptation au contexte (chapitre 5) que les applications issues du domaine de l'informatique pervasive doivent de manière intrinsèque être adaptée dynamiquement au contexte de l'utilisation et à ses changements. L'utilisation de tâches et de sous-tâches pour résoudre des problèmes est aussi utilisée dans le domaine de l'informatique pervasive. [Weiser, 1991] envisage de considérer les applications pervasives comme un ensemble de tâches et de sous-tâches d'utilisateur, et non comme une collection de programmes de calcul. Des projets issus de ce domaine [Banavar *et al.*, 2000] [Garlan *et al.*, 2002] [Rudolph, 2001], structurent les applications pervasives, qui sont vues comme des problèmes de composition de logiciels, en termes de tâches et de sous-tâches. L'utilisation de méthodes de résolution de problèmes par modèle de tâches dans un modèle de composition de services Web permet de :

- **Répondre dynamiquement aux changements sous-jacents au domaine de l'informatique pervasive,** par conséquent intégrer le concept d'adaptation au contexte d'utilisation à la définition de la composition de services Web. En effet, chaque élément du contexte à adapter peut être vu comme une tâche à résoudre. Lors de la résolution du problème (*i.e.* l'exécution de la composition de services), si le contexte change, une nouvelle tâche est appelée selon les changements des valeurs du contexte.

La section qui suit décrit le modèle de composition que nous proposons qui repose sur les méthodes de résolution de problèmes par modèle de tâches pour définir les services intervenant dans la composition.

8.4 Structure du modèle de composition ProbCWS

À ce jour, aucun langage de composition de services Web ne couvre l'ensemble des aspects impliqués par la conception des applications adaptées au contexte d'utilisation à partir de services Web indépendants et autonomes. Cette conception doit en parallèle répondre aux problématiques sous-jacentes de la composition de services Web – contrôle de l'exécution, évolutivité de la composition, et compensation de services.

Notre proposition de modèle de composition de services Web, ProbCWS (*Problem-solving for Composition of Web Services*), est une base à la mise en œuvre de la composition de services Web par une plate-forme (*cf.* chapitre 11) qui répond à l'ensemble des objectifs identifiés par notre problématique de travail. ProbCWS propose un moyen de définir l'ensemble des services Web nécessaires à la composition, à l'aide d'une méthode de résolution de problème par modèle de tâches (étudiée dans la section précédente).

En utilisant une telle méthode, le modèle de composition de services Web doit reposer sur l'architecture sous-jacente à cette dernière (*cf.* section 8.4.1). Nous décrivons tout d'abord

l'opérationnalisation de cette architecture dans le cadre de ProbCWS. Ensuite nous illustrons, les acteurs intervenant dans la mise en œuvre du modèle.

8.4.1 Architecture du modèle de composition ProbCWS

L'architecture du modèle de composition ProbCWS est intrinsèque à celle des applications utilisées lors de la mise en œuvre des méthodes de résolution de problèmes (*cf.* Figure 8.2). Une telle architecture doit être composée : d'une base de connaissances des problèmes, d'un moteur d'inférence et d'une base de connaissances du domaine.

La base de connaissances des problèmes. La base de connaissances des problèmes contient les définitions des problèmes et des stratégies.

Le moteur d'inférence. Le moteur d'inférence réalise la résolution de problèmes définis dans la base de connaissances des problèmes. Ce moteur repose sur des langages ou systèmes de résolutions de problèmes, tels que le système SCARP (Système Coopératifs d'Aide à la Résolution de Problèmes) [Willamowski, 1994], la méthodologie CommonKADS [Schreiber *et al.*, 1994] mise en œuvre dans le système KADS [Breuker *et al.*, 1988] [Wielenga *et al.*, 1986] ou le langage de résolution de problème AROMTasks [Genoud *et al.*, 2005] mis en œuvre au sein du système de représentation de connaissances par objets AROM [Page *et al.*, 2001]. Le choix du moteur d'inférence dans notre travail est déterminé dans le chapitre 9.

La base de connaissances du domaine. La base de connaissances du domaine décrit les entités primaires dont le moteur d'inférence a besoin afin de réaliser la résolution de problèmes décrit par la base de connaissances des problèmes. Dans notre travail, la base de connaissances du domaine rassemble :

- Les informations concernant **la requête du client en termes de composition de services Web**, en vue de réaliser une application adaptée au contexte d'utilisation à base de services Web ;
- Les **services Web** disponibles qui peuvent être invoqués lors de la mise en œuvre de telles applications.

Les représentations des services Web et de la requête de services, exprimées en WSR-Model décrit dans le chapitre précédent, forment la base de connaissances du domaine.

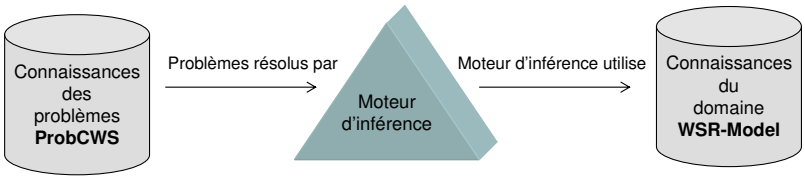


Figure 8.4 Architecture pour la mise en œuvre du modèle de composition de services Web ProbCWS.

L'architecture qui découle de notre modèle de composition de services Web ProbCWS est illustrée dans la Figure 8.4.

8.4.2 Cas d'utilisation du modèle de composition ProbCWS

Le modèle de composition de services Web ProbCWS a pour objectif principal de proposer aux clients un moyen de définir une composition de services Web en mettant en œuvre deux aspects. Ces deux aspects sont intégrer à la définition de la composition et réside en la mise en œuvre : (1) d'une

forme d'évolutivité de la composition ; (2) de l'adaptation au contexte d'utilisation (*cf.* cas d'utilisation UML illustré par la Figure 8.5).

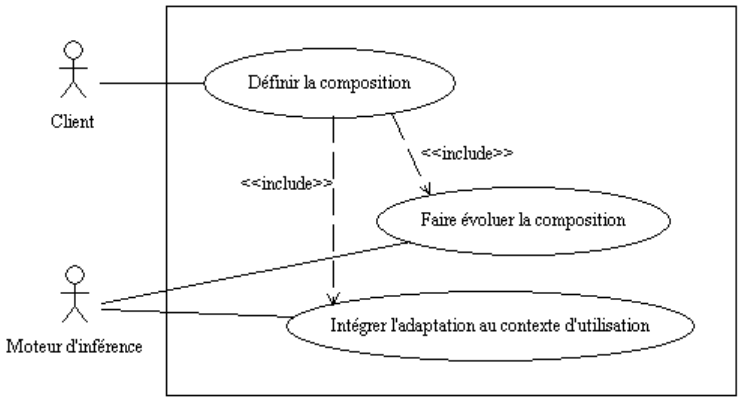


Figure 8.5 Cas d'utilisation du modèle de composition de services ProbCWS.

Définir la composition. Le client est l'acteur de la définition de la composition. Cet acteur, en suivant le modèle ProbCWS, définit le problème à résoudre (la tâche principale) et sa stratégie de décomposition (les sous-tâches) jusqu'à atteindre un service Web (tâche élémentaire), une catégorie de services Web (tâche spécialisable), ou une composition pré-définis. La liste des services et des compositions disponibles peut être recherchée par le client dans les bases de connaissances (du domaine et des problèmes). Cette recherche repose sur l'ensemble des critères du modèle WSR-Model (*cf.* chapitre précédent). La définition de la composition avec ProbCWS est étudiée dans la section 8.5. Le cas d'utilisation `Définir composition` inclut deux autres cas d'utilisation (`Mettre en œuvre l'évolutivité de la composition` et `Intégrer l'adaptation au contexte d'utilisation`) – *cf.* Figure 8.5.

Mettre en œuvre l'évolutivité de la composition. Cette évolutivité de la composition, réalisée par le moteur d'inférence, est rendue possible lorsque le client définit comme composant de la composition :

- un service Web élémentaire. Dans ce cas, si ce dernier échoue, alors le moteur d'inférence appelle un service Web appartenant à la même catégorie que le premier service (mise en œuvre de la **technique d'adaptation par compensation** – *cf.* section 8.7.1).
- une catégorie de services plutôt qu'un service Web. Alors le moteur d'inférence peut appeler un autre service Web appartenant à cette catégorie si le premier service échoue (mise en œuvre de la **technique d'adaptation par sélection** – *cf.* section 8.7.1).
- un ensemble de compositions existantes. Dans ce cas, si un service de la composition échoue alors le moteur invoque un service de la même catégorie, ou bien résout une autre composition de services appartenant à la même catégorie que celle définie par le client (mise en œuvre de la **technique d'adaptation par sélection de compositions** – *cf.* section 8.7.1).

Intégrer l'adaptation au contexte d'utilisation. Lors de la définition de la composition par le client, l'adaptation au contexte d'utilisation peut être intégrée (suivant le package *Use Context* du modèle de représentation de services Web – *cf.* section 7.2.6). La mise en œuvre de l'adaptation au contexte d'utilisation, par le biais de différentes formes d'adaptation (adaptation de la composition, du résultat, et de la demande), et techniques d'adaptation (par sélection,

composition, et compensation), est réalisée par le moteur d'inférence. La description de la mise en œuvre de l'adaptation au contexte d'utilisation est l'objet de la section 8.7.

L'ensemble des tâches élémentaires (dans notre travail, ces tâches sont des services Web), issues de la résolution du problème ainsi que la description de leur enchaînement, représentent le plan d'actions. L'exécution, la surveillance, et la compensation de services sont réalisées par le moteur de résolution de problèmes choisi. La mise en œuvre du moteur d'exécution de la composition de services ainsi que ses apports spécifiques sont décrits dans le chapitre 11.

8.5 Définition des concepts de base de la composition de services Web avec ProbCWS

Nous venons de présenter la méthode de résolution de problèmes par modèle de tâches sur laquelle ProbCWS s'appuie ainsi que les besoins auxquels ce modèle doit répondre. Nous décrivons, dans cette section, les concepts de base du modèle qui permettent de définir la composition à l'aide de ProbCWS.

Ce modèle de composition de services Web repose principalement sur les concepts de tâche, de définition de problème (*i.e.* la tâche principale), et sa résolution. Nous décrivons dans un premier temps les définitions de ces concepts dans ProbCWS. Ensuite, l'expression d'une résolution dans son intégralité ainsi qu'une illustration de cette résolution sont présentées.

Nous avons choisi de représenter le modèle de composition ProbCWS via le formalisme de représentation par objets UML.

8.5.1 Définition de la tâche dans ProbCWS

Afin de représenter les tâches permettant de définir et résoudre les problèmes (*i.e.* définir la composition), nous intégrons au modèle ProbCWS le package *Task*. Nous considérons une tâche comme étant une entité représentant, soit une tâche principale (le problème à résoudre), soit une tâche élémentaire (les moyens de résoudre le problème).

La Figure 8.6 illustre un extrait du package *Task* : la classe mère *Task* représente une tâche, spécialisée en deux sous-classes (les classes *MainTask* et *ElementaryTask*) représentant respectivement les tâches principales et les tâches élémentaires.

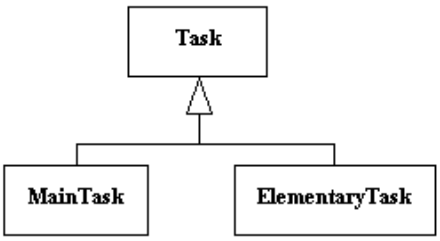


Figure 8.6 Extrait du package *Task*.

L'objectif de la définition d'une tâche est double :

- si elle consiste à définir une tâche principale, elle permet aux clients de **formaliser la demande de composition** ;
- si elle consiste à définir une tâche élémentaire, elle permet de **décrire la résolution du problème** (*i.e.* la composition).

La suite de cette section décrit tout d'abord la signature d'une tâche, puis les différents types de tâches que ProbCWS permet de décrire.

8.5.1.1 Signature d'une tâche

La signature d'une tâche a pour but d'enrichir la définition de la tâche (par conséquent la formalisation de la demande du client et la description de la résolution du problème) et est composée de trois éléments : ses arguments, le domaine d'application, et le contexte d'utilisation.

La définition d'une tâche contient tout d'abord ses **arguments** (les paramètres d'entrée et de sortie). Ceci permet de connaître les types et les noms des paramètres d'entrée et de sortie de la tâche.

D'après [Peer, 2005], la formalisation des buts dans un modèle de planification, dans le contexte de la composition de services Web, doit contenir des informations supplémentaires telles que : les stratégies pour pallier le non déterminisme, la sécurité des propriétés, la distinction entre les informations et la réalisation des buts, les préférences de l'utilisateur, etc. Afin de prendre en compte cet aspect d'enrichissement de la définition de la composition, nous ajoutons à la représentation d'une tâche la possibilité d'exprimer :

- le **domaine d'application** auquel le client souhaite que la composition appartienne et,
- le **contexte d'utilisation** auquel le client souhaite que la composition soit adaptée.

Afin de prendre en compte ces aspects (arguments, domaine d'application, et contexte d'utilisation), nous réalisons des associations entre la classe *Task* du package *Task* et entre différentes classes des packages *Functional Profile*, *Application Domain* et *Use Context* issus du modèle de représentation de services WSR-Model (*cf.* chapitre 7) – *cf.* Figure 8.7.

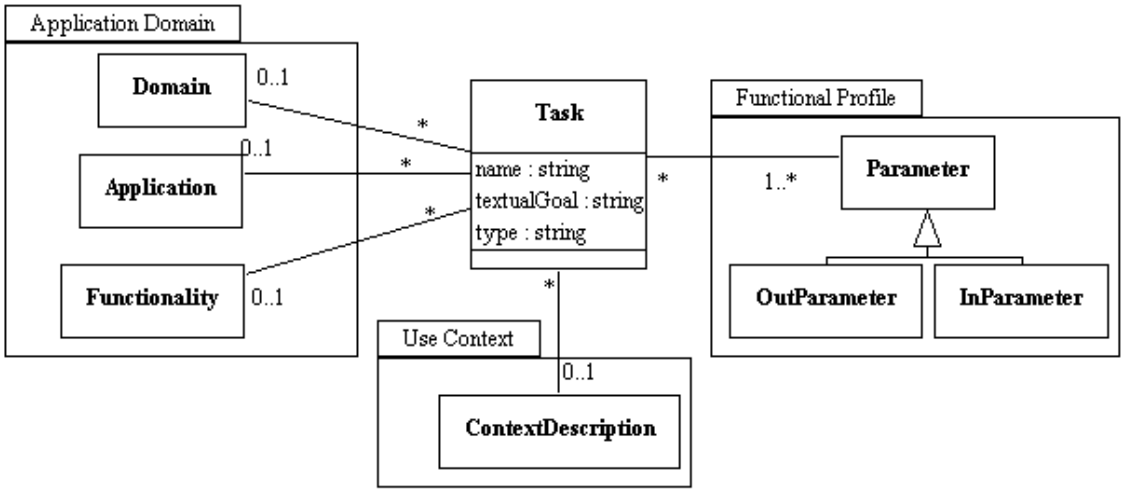


Figure 8.7 Diagramme de classes représentant la signature d'une tâche via les relations entre la classe *Task* du package *Task* et d'autres classes issues des packages *Application Domain*, *Functional Profile*, et *Use Context*.

Relation entre la classe *Task* et le package *Functional Profile*. La relation entre la classe *Task* et la classe *Parameter* du package *Functional Profile* permet de spécifier les arguments (paramètres d'entrée et de sortie) de la tâche. La définition des arguments est indispensable à la représentation d'une tâche.

Relation entre la classe *Task* et le package *Application Domain*. Les relations entre la classe *Task* et les classes du package *Application Domain* permettent d'ajouter à la définition d'une

tâche, le domaine d'application (avec la classe *Domain*), l'application (avec la classe *Application*), et la fonctionnalité (avec la classe *Functionality*) auxquels la tâche est associée.

Relation entre la classe *Task* et le package *Use Context*. La relation entre la classe *Task* et la classe *ContextDescription* du package *Use Context* permet de spécifier à quel contexte d'utilisation la tâche est adaptée.

Avec cette formalisation de la signature d'une tâche, ProbCWS permet d'ajouter la prise en compte du domaine d'application et de l'adaptation au contexte d'utilisation tant lors de l'expression du problème que de sa résolution.

Prenons l'exemple d'une tâche principale, nommée *myStock* (valeur de l'attribut *name* de la classe *Task*), dont le but est de *renvoyer le prix d'une action* (valeur de l'attribut *textual-goal* de la classe *Task*). Cette tâche principale appartient au domaine du *e-business* (association avec une instance de la classe *Domain* dont la valeur est *e-business*) et est adaptée à l'heure de l'ouverture de la bourse de New York (association avec une instance de la classe *DescriptionContext*, elle-même associée à une instance de la classe *Time* dont la valeur de l'attribut *time* est connue – *cf.* le package *Use Context* de WSR-Model, section 7.2.6).

8.5.1.2 Types de tâche

Dans ProbCWS, nous définissons trois types de tâche (représentés par l'attribut *type* de la classe *Task* – *cf.* Figure 8.7) qui permettent de mettre en œuvre une résolution de problème adaptée au contexte d'utilisation : des **tâches classiques**, des **tâches adaptées**, et des **tâches d'adaptation**. Ces types de tâches sont inférés et diffèrent selon les valeurs des éléments *domaine d'application* et *contexte d'utilisation* de la signature de la tâche.

Les tâches classiques. Une tâche classique est une tâche qui ne prend pas en compte le contexte d'utilisation. Par conséquent ce type de tâche ne possède pas dans sa signature l'élément décrivant le contexte d'utilisation. Les tâches classiques sont utilisées :

- soit pour décrire un problème pour lequel le client ne désire pas d'adaptation,
- soit pour résoudre un problème pour lequel il n'y a pas d'adaptation au contexte d'utilisation,
- soit pour résoudre une partie d'un problème qui doit être adapté à un contexte d'utilisation, mais pour lequel aucune tâche dans la base de connaissances ne répond à la demande du client et est adaptée au contexte d'utilisation.

Les tâches adaptées. Une tâche adaptée est une tâche qui répond à la demande (ou une partie) du client en intégrant la mise en œuvre de l'adaptation au contexte d'utilisation (ou une partie) demandée par le client. Par conséquent, ce type de tâche possède obligatoirement dans sa signature les arguments et la description du contexte d'utilisation. Les tâches adaptées sont utilisées :

- soit pour décrire un problème pour lequel le client souhaite intégrer dans sa résolution l'adaptation à un contexte d'utilisation,
- soit pour résoudre un problème qui répond à l'intégralité de la signature de la tâche (fonctionnalité et adaptation au contexte d'utilisation).

Les tâches d'adaptation. Une tâche d'adaptation est une tâche qui met en œuvre l'adaptation au contexte d'utilisation (ou à une partie du contexte d'utilisation) mais qui ne répond pas directement à la demande du client ou à la décomposition de la demande du client. La signature de la tâche d'adaptation ne possède pas de description du domaine d'application et ses arguments ne sont pas ceux de la demande du client ou de sa décomposition (si cela est le cas, il s'agit d'une tâche adaptée).

Les tâches d'adaptation sont utilisées pour mettre en œuvre l'adaptation au contexte d'utilisation, en partie ou dans son intégralité, lorsqu'il n'existe pas de tâches adaptées correspondant à la fonctionnalité et au contexte d'utilisation demandés.

Nous venons d'étudier le concept de la tâche dans ProbCWS. Ce qui suit définit de manière plus spécifique le concept de tâche principale.

8.5.2 Définition de la tâche principale dans ProbCWS

La tâche principale permet de définir le problème à résoudre du client en termes de composition de services Web. À l'instar d'une tâche, sa définition est composée d'une signature décrite par ses arguments, le domaine d'application auquel le client souhaite que la composition appartienne, ainsi que le contexte d'utilisation pour lequel le client souhaite que la composition soit adaptée.

Afin de prendre en compte les spécificités d'une tâche principale au sein du modèle ProbCWS, nous ajoutons à ce modèle (*cf.* Figure 8.8) :

Le package *Composition* contenant deux classes (les classes *Composition* et *CompositionCategory*) qui représentent respectivement une composition et la (ou les) catégorie(s) à laquelle (auxquelles) la composition est associée (association *represents*).

L'association entre la tâche principale (classe *MainTask*) **et la composition** (classe *Composition*) afin de représenter la composition pour laquelle la tâche principale est définie.

L'association entre la composition (classe *Composition*) **et la description du contexte** (classe *ContextDescription* du package *Use Context*) afin de représenter le contexte pour lequel la composition est adaptée.

L'association entre la catégorie de compositions (classe *CompositionCategory*) **et le domaine d'application** (classe *Domain* du package *Application Domain*) afin de représenter le domaine auquel appartient la catégorie de compositions.

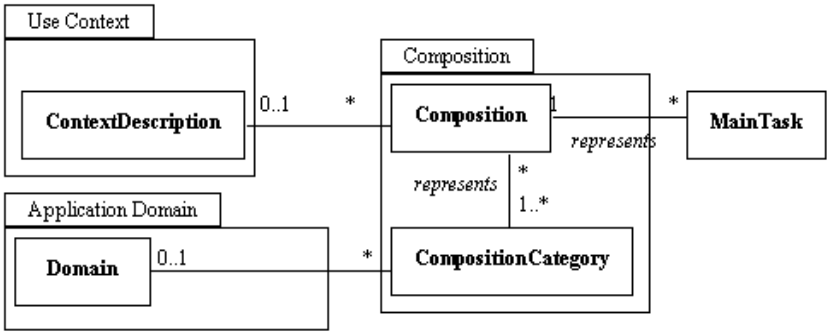


Figure 8.8 Diagramme de classes représentant la classe *MainTask* et le lien avec le package *Composition*.

Ce qui suit décrit les concepts inhérents à la résolution de problème (*i.e.* la tâche principale) dans ProbCWS.

8.5.3 Définition de la résolution de problèmes dans ProbCWS

Dans notre travail, le problème (ou tâche) principal(e) est la demande du client en termes de composition de services Web, via ses arguments, son domaine d'application, et son contexte

d'utilisation, *cf.* section 8.5.1.1. Deux concepts sont mis en jeu afin de résoudre une tâche principale : la *tâche élémentaire* et la *stratégie.*

8.5.3.1 La tâche élémentaire

Une tâche élémentaire a pour but de résoudre le problème, formalisé par une tâche principale.

Puisque nous nous basons sur la méthode de résolution de problèmes à base de tâches, cette résolution, à laquelle participe les tâches élémentaires, consiste à décomposer le problème en problèmes d'une granularité plus fine. Dans ProbCWS, nous formalisons ceci par une relation de composition entre les classes *MainTask* et *ElementaryTask* (*cf.* Figure 8.9).

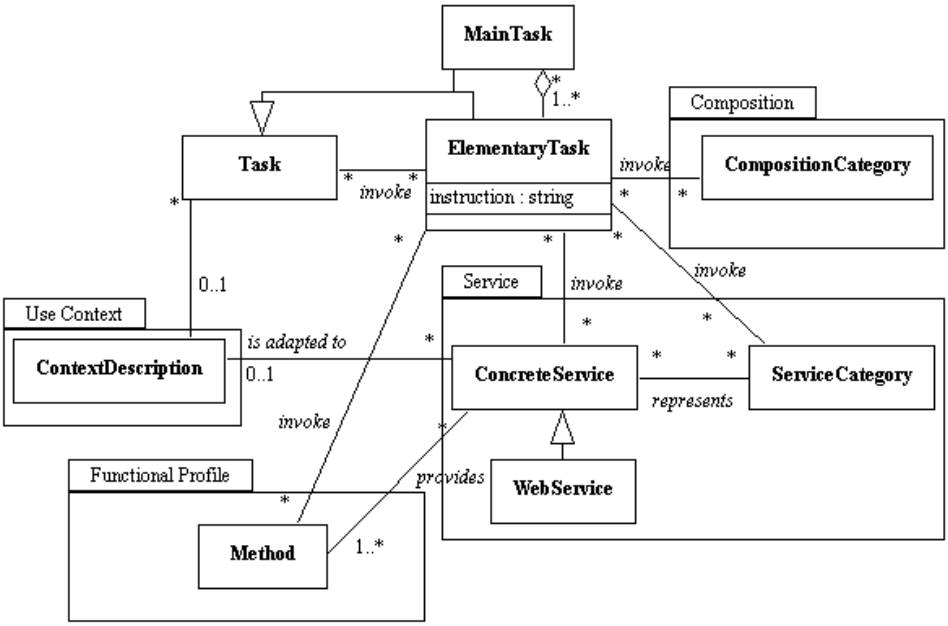


Figure 8.9 Diagramme de classes représentant les liens entre la classe *ElementaryTask* du package *Task* et d'autres classes issues des packages *Task*, *Service*, et *Use Context*.

Une tâche élémentaire permet de mettre en œuvre la résolution d'une tâche principale par l'intermédiaire d'**instructions** (attribut *instruction* de la classe *ElementaryTask*). Les instructions définies dans la tâche élémentaire peuvent être de trois types : affectation, appel, et instruction de contrôle.

L'affectation. Cette instruction permet de modifier l'état, soit d'une variable locale, soit d'un argument de la tâche élémentaire (défini dans la signature de cette dernière). L'affectation de la variable est une expression impliquant d'autres variables, des opérateurs mathématiques, ou des appels à des tâches ou des services Web (*cf.* instruction d'appel ci-dessous).

L'appel. Cette instruction permet de modifier l'état d'une variable en invoquant :

- **une autre tâche** (principale ou tâche élémentaire) répertoriée dans la base de connaissances des problèmes. Ce type d'appel est représenté dans le modèle par l'association *invoke* entre les classes *ElementaryTask* et *Task* (*cf.* Figure 8.9).
- **une catégorie de compositions**, répertoriée dans la base de connaissances du domaine. Ce type d'appel est représenté par l'association *invoke* entre les classes *ElementaryTask* et *CompositionCategory* (*cf.* Figure 8.9). Ceci a pour conséquence de mettre en œuvre la résolution d'une autre tâche principale dont le client ne connaît pas le nom.

- **une catégorie de services**, répertoriée dans la base de connaissances du domaine. Ce type d'appel est représenté par l'association *invoke* entre les classes *ElementaryTask* et *ServiceCategory* (*cf.* Figure 8.9). Ceci est une résolution de plus bas niveau que les deux précédents types d'appel (tâche et catégorie de compositions). L'appel d'une catégorie de services est utile lorsque le client qui définit la composition ne connaît pas au préalable un service concret qui peut résoudre le problème, mais en connaît la catégorie.
- **une méthode particulière d'un service**, répertoriée dans la base de connaissances du domaine. Ce type d'appel est représenté par l'association *invoke* entre les classes *ElementaryTask* et *Method* (*cf.* Figure 8.9) et est l'appel du plus bas niveau de granularité dans la résolution d'un problème ProbCWS. Ce type d'invocation permet d'exécuter la composition de services. Le service concret, qui fournit la méthode à appeler, peut être implémenté par un service Web. Cette invocation permet aussi d'implémenter les différents types de tâches (tâches classiques, tâches adaptées, et tâches d'adaptation) définies plus haut (*cf.* section 8.5.1.2), par les différents types de services (respectivement services classiques, services adaptés, et services d'adaptation) définis dans le chapitre précédent (*cf.* section 7.4). Nous associons l'appel d'une méthode d'un service à l'appel d'un service (*cf.* association *invoke* entre les classes *ElementaryTask* et *ConcreteService*) afin de mettre en évidence, au niveau modèle, le fait qu'une tâche élémentaire est associée également au service dont la méthode invoquée dépend. Ceci est utile par la suite afin de mettre en œuvre la technique d'adaptation par compensation (recherche de services équivalents – même catégorie de services, qui possèdent des méthodes semblables – mêmes paramètres d'entrée et de sortie).

Les instructions de contrôle. Les instructions d'une tâche élémentaire peuvent être exécutées, soit de manière conditionnelle (à l'aide, par exemple, des instructions `if`, ou `if then else`), soit de manière itérative (à l'aide, par exemple, de l'instruction `while then`). Les instructions conditionnelles permettent d'implémenter les tâches spécialisables (*cf.* section 8.3.1.2), tandis que les instructions itératives permettent d'implémenter les tâches composables (*cf.* section 8.3.1.2).

8.5.3.2 La stratégie

Une stratégie permet d'associer aux tâches principales les tâches élémentaires qui mettent en œuvre la résolution de problèmes. Dans ProbCWS, la stratégie est représentée par la classe *Strategy*, associée à la relation de composition entre les classes *MainTask* et *ElementaryTask* (*cf.* Figure 8.10).

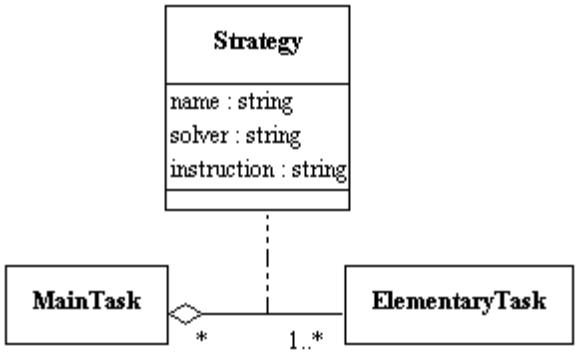


Figure 8.10 Extrait du package *Task* représentant la stratégie.

Une stratégie est identifiée par son nom (attribut *name* de la classe *Strategy*). La description d'une stratégie est composée de deux éléments : le lien vers la résolution du problème (attribut *solver* de la classe *Strategy*) et des instructions (attribut *instruction* de la classe *Strategy*).

Le lien vers la résolution du problème. L'identification de la (ou des) tâche(s) élémentaire(s) permettant de résoudre la tâche principale sont définies dans l'attribut *solver*. La tâche principale à résoudre et la (ou les) tâches élémentaires qui réalise(nt) la résolution doivent posséder les mêmes arguments (*i.e.* même paramètres d'entrée et de sortie).

Les instructions d'une stratégie. La description de l'instruction d'une stratégie permet d'intégrer si besoin des instructions particulières à exécuter. Les instructions définies dans une stratégie permettent de mettre en œuvre :

- les instructions de contrôle telles que les conditions et les itérations,
- l'appel de tâche(s) élémentaire(s), identifiée(s) par l'élément `solver`, permettant de résoudre la tâche principale.

8.5.4 Expression d'une résolution de problèmes

Le modèle de composition de services Web ProbCWS permet, à l'aide de méthodes de résolution de problèmes par modèle de tâches, de définir l'ensemble des services Web nécessaires à la réalisation d'une composition de services Web.

Dans l'expression de la résolution de problèmes cinq entités, nécessaires pour la mise en œuvre de la résolution, sont décrites :

La base de connaissances des problèmes. Cette base, utilisée lors de la résolution de problèmes, est identifiée par son nom.

La base de connaissances du domaine. Cette base, appelée lors de l'exécution de la résolution de problèmes, est identifiée par son nom.

Les tâches principales. L'ensemble des définitions des tâches principales décrivant le problème et sa résolution est décrit.

Les tâches élémentaires. L'ensemble des définitions des tâches élémentaires permettant de mettre en œuvre l'exécution de la résolution de problèmes est décrit.

Les stratégies. Les stratégies permettent de faire le lien entre les tâches principales (le problème) et les tâches élémentaires (les résolutions du problème). Ces stratégies sont le dernier élément de la définition de la résolution de problèmes à être décrit.

L'ensemble des tâches élémentaires invoquant des services, issu de la résolution du problème (la demande du client), ainsi que la description de leur enchaînement (définies à partir des instructions des tâches élémentaires), représentent le plan d'actions. L'exécution, la surveillance, et la compensation de services sont réalisées par le moteur de résolution de problèmes choisit (*cf.* chapitre 9). La mise en œuvre du moteur d'exécution de la composition de services ainsi que ses apports spécifiques sont décrits dans le chapitre 11.

Ce qui suit illustre l'utilisation du modèle ProbCWS avec un exemple concret de composition de services Web.

8.5.5 Illustration de l'utilisation du modèle de composition de services Web ProbCWS

L'exemple qui sert d'illustration à l'utilisation du modèle ProbCWS est la résolution du problème *myMeteo*, qui est une composition de trois services Web[67] nommés *geoIP*, *globalWeather* et *xml2dial.* Ce problème a pour objectif de fournir au client une application de service météorologique ne nécessitant pas le paramétrage de la localisation.

L'étude de l'exemple commence par une vue d'ensemble de la résolution du problème. Dans cet exemple, la résolution du problème est décomposée en quatre triplets (*tâche principale*, *tâche élémentaire*, *stratégie*). Nous consacrons un paragraphe à la description de chacun de ces triplets :

- Le premier exprime le problème à résoudre défini par le client.
- Le second triplet permet de convertir le paramètre d'entrée afin d'exécuter la fonctionnalité demandée par le client.
- Le troisième triplet exprime le sous-problème qui résout une partie du problème principal (exécution de la fonctionnalité demandée par le client).
- Le dernier triplet composant la résolution du problème est la conversion de la sortie de la fonctionnalité afin de correspondre à la demande du client.

La composition de services Web proprement dite permet de mettre en œuvre cette résolution de problèmes reposant sur trois services Web. Ces derniers existent au préalable dans la base de connaissances du domaine, ayant été publiés par leur fournisseur. Par défaut, à chacune des méthodes des services Web publiés est associé un triplet : une tâche principale qui représente la fonctionnalité du service (*i.e.* le problème à résoudre) ; une stratégie qui associe la tâche principale à la tâche élémentaire ; et une tâche élémentaire qui permet d'invoquer le service. Le client, à l'origine de la résolution du problème *myMeteo*, peut faire appel, dans la définition de la composition, aux trois tâches principales disponibles dans la base de connaissances des problèmes.

Notons que les instructions des tâches élémentaires et des stratégies sont données en pseudo-code dans la suite du chapitre. Le chapitre suivant décrit un moyen d'opérationnaliser le modèle et ainsi apporte une syntaxe plus précise à ce dernier.

8.5.5.1 Vue d'ensemble

Le problème à résoudre issu de la demande de l'utilisateur est représenté par la tâche principale *myMeteo* et contient un contexte d'utilisation (composé des éléments *localization* et *mobileDevice*) pour lequel le client souhaite une réponse adaptée. Une stratégie associe à cette tâche principale une tâche élémentaire nommée *compositionTask*. Cette tâche élémentaire fait appel à trois autres tâches principales :

- La tâche principale, nommée *ipToTown*, est associée par une stratégie à une tâche élémentaire *geoIP* qui appelle un premier service Web qui permet de connaître la localisation de l'utilisateur (sa ville) selon son IP.
- La tâche principale, nommée *weather*, intègre une partie du contexte d'utilisation (seulement l'élément *localization*). Cette tâche principale est associée par l'intermédiaire d'une stratégie à une tâche élémentaire nommée *globalWeather*. Cette tâche élémentaire appelle un service (le second dans la composition) qui, à partir de la localisation de l'utilisateur, retourne les conditions météorologiques courantes.

[67] La composition des deux premiers services Web (*geoIp* et *globalWeather*) a servi d'exemple au chapitre 14 consacré à l'étude de la composition de services Web.

- La tâche principale, nommée *xmlToDial*, est associée par une stratégie à une tâche élémentaire nommée *xml2Dial*. Cette tâche élémentaire appelle un service (le troisième et dernier dans la composition) qui convertit la sortie du précédent service Web en format DIAL [Smith, 2007], format souhaité pa le client.

L'expression de base de la résolution du problème *myMeteo* est composée de :

- la base de connaissances des problèmes que nous nommons *myProblemKB* ;
- la base de connaissances du domaine que nous nommons *myKB* ;
- les trois triplets de définition de la composition représentés par une tâche principale, une tâche élémentaire et une stratégie : (*myMeteo*, *compositionTask*, *myMeteo*), (*ipToTown*, *geoIP*, *ipToTown*), (*weather*, *globalWeather*, *weather*), et (*xmlToDial*, *xml2dial*, *xmlToDial*). Chacune des tâches principales est associée à la tâche élémentaire par le biais de la stratégie.

Le paragraphe suivant décrit le premier triplet (*myMeteo*, *compositionTask*, *myMeteo*) représentant le problème principal à résoudre.

8.5.5.2 Problème à résoudre

La première tâche principale du problème *myMeteo* représente la demande du client dans son intégralité et est définie par ce même acteur. Cette tâche principale est décrite par son identification, ses arguments et le contexte d'utilisation auquel le résultat doit être adapté.

L'identification de la tâche principale. Cette tâche principale est identifiée par son nom (*myMeteo*) et par un texte descriptif (attribut *textual-goal* de la classe *Task*).

Les arguments de la tâche principale. Les types des paramètres d'entrée et de sortie désirés par le client sont un entier comme paramètre d'entrée et un fichier de type DIAL comme paramètre de sortie. Le paramètre d'entrée désigne le numéro IP de la machine de l'utilisateur final.

Le contexte d'utilisation pour lequel le client attend une adaptation. Dans cet exemple, le client attend de l'adaptation à un contexte d'utilisation composé de deux éléments : la ville et la taille de l'écran. La définition de ce contexte d'utilisation est permise via le package *UseContext* de WSR-Model.

La tâche élémentaire, nommée *compositionTask*, possède les mêmes types d'arguments (un paramètre d'entrée, `in`, de type entier, et un paramètre de sortie, `out`, de type DIAL) et le même contexte d'utilisation (les attributs *town* de la classe *Localization* et *screenSize* de la classe *MobileDevice*) que la tâche principale *myMeteo*. Cette tâche élémentaire, associée au problème à résoudre (la tâche principale *myMeteo*), permet de définir les enchaînements entre les tâches à invoquer. Ces enchaînements sont décrits dans les instructions de la tâche élémentaire (*cf.* Figure 8.11). Dans cet exemple, l'enchaînement est simple étant donné qu'il récupère les sorties des premières sous-tâches afin d'en établir les entrées des sous-tâches suivantes.

```
1  body
2      var1        string
3      var2        XML
4      var1    =   ipToTown(in)
5      var2    =   weather(var1)
6      out     =   xmlToDial(var2)
7  endbody
```

Figure 8.11 Instructions de la tâche élémentaire *compositionTask*.

Le paramètre d'entrée `in` de la tâche *compositionTask* devient le paramètre d'entrée de la tâche `ipToTown` (ligne 4). Le paramètre de sortie de `ipToTown`, défini comme la variable locale `var1` (ligne 4), devient le paramètre d'entrée de la tâche `weather` (ligne 5). Le paramètre de sortie de `weather`, défini comme la variable locale `var2` (ligne 5), devient le paramètre d'entrée de la tâche `xmlToDial` (ligne 6). Le paramètre de sortie de `xmlToDial` (ligne 6) est le paramètre de sortie `out` de `compositionTask`.

La stratégie *myMeteo* associe la tâche principale du même nom à la tâche élémentaire *compositionTask* par l'intermédiaire de l'attribut *solver* auquel la valeur *compositionTask* est donnée. L'instruction de la stratégie consiste à retourner l'exécution de la tâche (*cf.* Figure 8.12).

```
1  instruction
2      return "compositionTask"
3  endinstruction
```

Figure 8.12 Instructions de la stratégie myMeteo.

Les triplets décrits par la suite existent déjà dans la base de connaissances des problèmes. Le client à l'origine de la résolution du problème *myMeteo* n'a eu à sa charge que la définition de la tâche principale *myMeteo*, de la stratégie *myMeteo*, et de la tâche élémentaire *compositionTask* qui viennent d'être décrites.

Le paragraphe suivant décrit le second triplet (tâche principale, tâche élémentaire, stratégie) qui permet d'exécuter la résolution du problème *myMeteo*.

8.5.5.3 Sous-problème de conversion du paramètre d'entrée

Le second triplet (tâche principale, tâche élémentaire, stratégie) permet de définir la première partie du sous-problème qui représente le fait de convertir le paramètre d'entrée, le numéro IP de la machine de l'utilisateur final, dans la ville de l'utilisateur.

La tâche principale, nommée *ipToTown*, a pour paramètre d'entrée un entier et pour paramètre de sortie une chaîne de caractères.

La tâche élémentaire, nommée *geoIP*, possède les mêmes arguments que la tâche principale (nommés `in` pour le paramètre d'entrée et `out` pour le paramètre de sortie). Les instructions de la tâche (*cf.* Figure 8.13) retournent le résultat du service Web `GeoIP` comme paramètre de sortie. Cette tâche est classique puisqu'elle ne prend pas en compte le contexte d'utilisation et fait appel directement à un service référencé dans la base de connaissances du domaine.

```
1  body
2      out = call GeoIP(in)
3  bodyend
```

Figure 8.13 Instructions de la tâche élémentaire *geoIP*.

La stratégie `ipToTown` associe la tâche principale du même nom à la tâche élémentaire `geoIP` (identifiée par l'élément *solver* dont la valeur est `geoIP`). L'instruction de cette stratégie consiste à retourner l'exécution de la tâche, à l'instar de la stratégie précédente (*cf.* Figure 8.12).

Le paragraphe suivant décrit le triplet (tâche principale, sous-tâche, stratégie) qui apporte une partie de l'adaptation au contexte d'utilisation demandée par le client.

8.5.5.4 Sous-problème de résolution

Le troisième triplet (tâche principale, sous-tâche, stratégie) renvoie les données météorologiques selon la ville de l'utilisateur.

La tâche principale, nommée *weather*, a pour paramètre d'entrée une chaîne de caractères et pour paramètre de sortie un fichier XML. Cette tâche est adaptée à la localisation de l'utilisateur et plus particulièrement à sa ville, via l'association à la description de contexte (classe *ContextDescription*), composée de l'attribut *town* de la classe *Localization*.

La tâche élémentaire, nommée *globalWeather*, possède les mêmes arguments que la tâche principale (nommés `in` pour le paramètre d'entrée et `out` pour le paramètre de sortie) et s'adapte au même contexte d'utilisation que celui défini dans la tâche principale (même association avec la même description du contexte). L'instruction de la tâche (*cf.* Figure 8.14) retourne le résultat du service Web `globalWeather` comme paramètre de sortie. Cette tâche est une *tâche adaptée* puisqu'elle répond à une décomposition de la demande du client et prend en compte une partie de l'adaptation au contexte d'utilisation. Cette tâche élémentaire fait appel directement à un service Web référencé dans la base de connaissances du domaine.

```
1  body
2      out = call globalWeather(in)
3  bodyend
```

Figure 8.14 Instructions de la tâche élémentaire *globalWeather*.

La stratégie *weather* associe la tâche principale du même nom à la tâche *globalWeather* (identifiée par l'attribut *solver* qui a pour valeur *globalWeather*). L'instruction de cette stratégie retourne l'exécution de cette tâche élémentaire.

Le paragraphe suivant décrit le triplet (tâche principale, tâche élémentaire, stratégie) qui permet de convertir le résultat de l'exécution de la résolution du problème afin qu'il soit adapté à l'intégralité du contexte d'utilisation désiré par le client.

8.5.5.5 Sous-problème de conversion du paramètre de sortie

Le dernier triplet (tâche principale, sous-tâche, stratégie) permet d'adapter le résultat de l'exécution du problème pour convenir à la demande du client (adaptation à la taille de l'écran de l'utilisateur).

La tâche principale, nommée *xmlToDial*, a pour paramètre d'entrée un fichier XML et pour paramètre de sortie un fichier DIAL.

La tâche élémentaire, nommée *xml2dial*, possède les mêmes arguments que la tâche principale (nommés `in` pour le paramètre d'entrée et `out` pour le paramètre de sortie) et est associée à la même description de contexte que la tâche principale. L'instruction de la tâche élémentaire (*cf.* Figure 8.15) retourne le résultat du service Web `XML2DIAL` comme paramètre de sortie de la tâche *xml2dial*. Cette tâche est une *tâche d'adaptation* puisqu'elle prend en compte le contexte d'utilisation sans être une tâche issue de la décomposition directe de la demande du client. Cette tâche fait appel directement à un service Web référencé dans la base de connaissances du domaine.

```
1  body
2      out = call XML2DIAL(in)
3  endbody
```

Figure 8.15 Instructions de la tâche *xml2dial*.

La stratégie *xmlToDial* associe la tâche principale du même nom à la tâche *xml2dial* (identifiée par l'élément *solver* dont la valeur est *xml2dial*). L'instruction de la stratégie retourne l'exécution de la tâche *xml2dial*.

L'ensemble des tâches et des stratégies décrites ici forme la définition de la résolution d'un problème qui permet de mettre en œuvre une composition de services Web via ProbCWS. Notons que la tâche principale décrite par le client est stockée dans la base de connaissances des problèmes. Ainsi elle pourra être réutilisée par d'autres clients dans d'autres résolutions de problèmes.

8.6 Arbre de résolution ProbCWS

Nous proposons une notation graphique de ProbCWS inspirée des arbres de résolution utilisés dans le domaine de la résolution de problèmes par modèle de tâches (*cf.* section 8.3.1.2).

À l'aide de l'arbre de résolution ProbCWS, nous pouvons illustrer de manière graphique les points clés de ce modèle de composition :

- Les différents concepts manipulés (tâche principale, tâche élémentaire, stratégie) et leurs liens.
- La composition de services Web par l'intermédiaire de la résolution de problèmes par modèle de tâches.
- Le flux de données.
- La possibilité de mettre en œuvre l'adaptation au contexte d'utilisation (étudiée dans la section suivante).

Nous donnons les éléments graphiques de chacun des concepts inhérents à ProbCWS par l'intermédiaire des représentations des tâches et de la résolution de problèmes. Puis nous donnons un exemple de représentation d'une composition avec l'arbre de résolution ProbCWS.

8.6.1 Représentation des tâches

L'arbre de résolution ProbCWS possède une notation distincte pour chacune des tâches : tâches principales et tâches élémentaires.

La **tâche principale** est illustrée par l'intermédiaire d'un losange. Nous représentons aussi les entrées (E) et sorties (S), arguments de la tâche principale (*cf.* Figure 8.16).

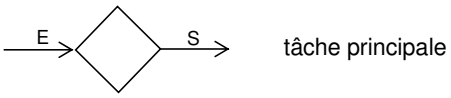


Figure 8.16 Notation graphique d'une tâche principale.

Une **tâche élémentaire** est représentée par un cercle. Selon le *type de la tâche*, le cercle prend une teinte différente : blanc pour une tâche classique, gris pour une tâche adaptée, et noir pour une tâche d'adaptation (*cf.* Figure 8.17 à gauche). Si la tâche élémentaire invoque un service concret, elle est représentée par deux cercles concentriques (*cf.* Figure 8.17 à droite). Dans ce dernier cas, les teintes varient selon le type de service invoqué (blanc pour un service classique, gris pour un service adapté, et noir pour un service d'adaptation).

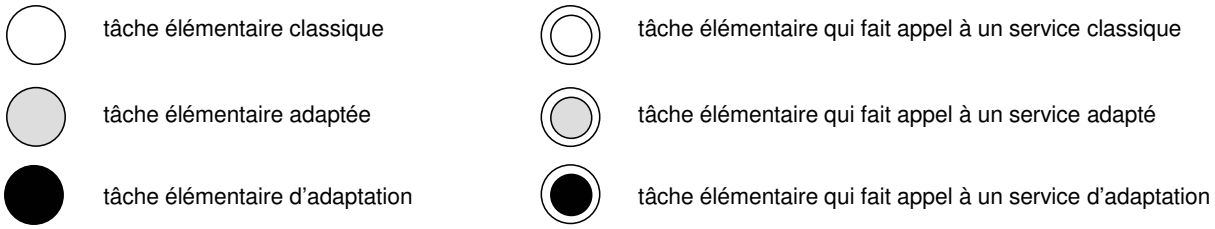


Figure 8.17 Notation graphique des tâches élémentaires.

La définition d'une tâche élémentaire inclut aussi les *instructions*. L'arbre de résolution ProbCWS distingue trois instructions : la décomposition, la spécialisation, et l'appel. Concernant les deux premiers types d'instructions, nous suivons les spécifications données par [Crampé, 1994] (*cf.* section 8.3.1.2) – Figure 8.18. Nous ajoutons la notation graphique d'un appel simple représenté par un trait vertical (*cf.* Figure 8.18). La *décomposition* permet de mettre en œuvre un enchaînement séquentiel entre les appels de tâches. La *spécialisation* permet d'introduire des alternatives en termes de tâches équivalentes.

Figure 8.18 Notation graphique des instructions des tâches élémentaires.

8.6.2 Représentation de la résolution de problèmes

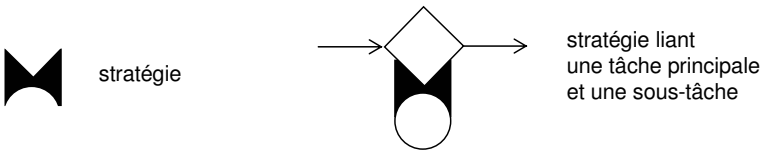


Figure 8.19 Notation graphique d'une stratégie.

La **stratégie** permet de lier la tâche principale et sa résolution (tâche élémentaire). C'est pourquoi nous avons choisi une représentation graphique permettant d'imbriquer un losange (tâche principale) et un cercle (tâche élémentaire) – *cf.* Figure 8.19.

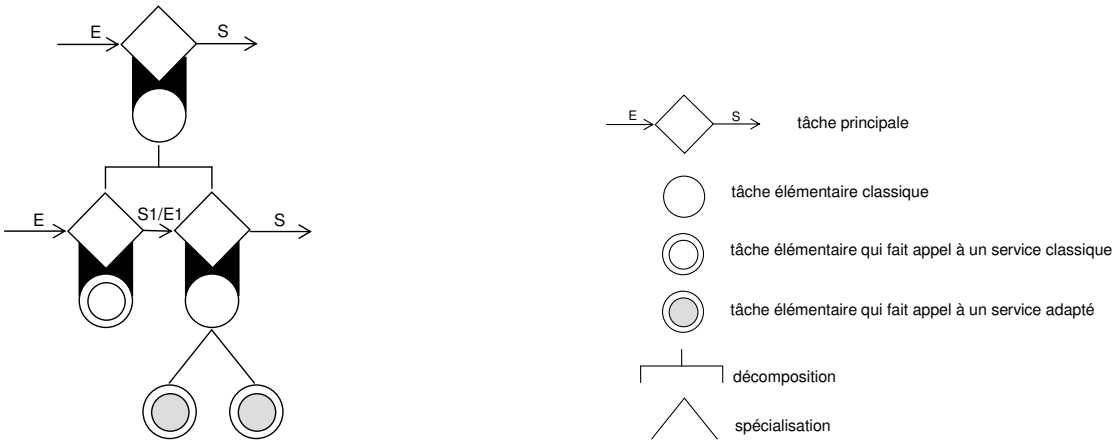


Figure 8.20 Exemple de l'illustration graphique d'une résolution de problèmes avec l'arbre de résolution ProbCWS.

La Figure 8.20 montre un exemple d'une résolution de problèmes illustrée avec l'arbre de résolution ProbCWS. Ce problème (tâche principale, racine de l'arbre) a pour argument une entrée E

et une sortie S, et peut être résolu par une tâche élémentaire. Cette tâche élémentaire est décomposable en deux tâches principales :

- La première tâche principale est résolue par une tâche élémentaire faisant appel à un service classique. L'entrée de cette tâche principale est l'entrée de la tâche principale racine (le problème de base à résoudre).
- La seconde tâche principale (qui prend pour paramètre d'entrée la sortie de la première tâche principale) est résolue par une tâche spécialisable en deux tâches élémentaires faisant appel à des services adaptés. La sortie de cette tâche principale est la sortie de la tâche principale racine.

8.6.3 Illustration d'une représentation d'une composition

La Figure 8.21 illustre une vue d'ensemble, à l'aide d'un arbre de résolution ProbCWS, du problème *myMeteo* (*cf.* section 8.5.5). La résolution de problèmes est décomposée en quatre tâches principales (la racine et trois nœuds) ainsi qu'en quatre tâches élémentaire (un nœud et trois feuilles).

Le problème à résoudre (racine de l'arbre) est représenté par la tâche principale *myMeteo* qui a *in* comme paramètre d'entrée de type string, et *out* comme paramètre de sortie de type DIAL. Une stratégie associe à cette tâche principale une tâche élémentaire nommée *compositionTask*.

La tâche élémentaire *compositionTask* est décomposable en trois tâches principales (*ipToTown*, *weather*, et *xmlToDial*) qui vont être appelées de manière séquentielle : les paramètres de sortie de l'une étant les paramètres d'entrée de la suivante. Les appels et les flux de données sont décrits via l'instruction de la tâche élémentaire *compositionTask.*

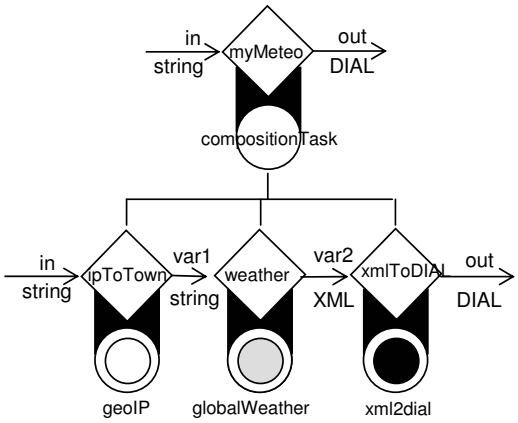


Figure 8.21 Représentation via l'arbre de résolution ProbCWS de la résolution du problème *myMeteo*.

La tâche principale *ipToTown* est associée par une stratégie à la tâche élémentaire *geoIP* (feuille de l'arbre) qui appelle un premier service concret (de type classique). Le paramètre d'entrée de l'appel de ce service est celui du problème à résoudre (*in*). Le paramètre *var1*, la sortie de l'appel de ce service, devient le paramètre d'entrée de la tâche principale suivante.

La tâche principale *weather* est associée par l'intermédiaire d'une stratégie à la tâche élémentaire *globalWeather*. Cette tâche élémentaire appelle un service concret de type adapté (le second dans la composition). Le paramètre d'entrée de l'appel de ce service est celui de la tâche principale *weather* (*var1*). Le paramètre *var2*, la sortie de l'appel de ce service, devient le paramètre d'entrée de la tâche principale suivante.

La tâche principale *xmlToDial* est associée par une stratégie à la tâche élémentaire *xml2Dial*. Cette tâche élémentaire appelle un service d'adaptation (le troisième et dernier de la composition). Le paramètre d'entrée de l'appel de ce service est celui de la tâche principale *xmlToDial* (*var2*). Le paramètre *out*, la sortie de l'appel de ce service, est le paramètre de sortie du problème à résoudre.

8.7 Vers l'adaptation au contexte d'utilisation dans ProbCWS

La mise en œuvre au sein du modèle ProbCWS de l'adaptation au contexte d'utilisation réside tout d'abord dans l'**enrichissement de la requête du client**.

Lors de la définition de la requête, le client exprime un ensemble d'éléments. Ces éléments définissent la signature du problème composée des paramètres d'entrée et de sortie désirés, du domaine d'application auquel doit appartenir la résolution du problème, et du contexte d'utilisation. Ce dernier élément permet au client de formaliser pour quel contexte d'utilisation il désire que la composition de services Web définie soit adaptée. Dans l'exemple du problème *myMeteo* (*cf.* section 8.5.5), le client souhaite que la composition soit adaptée au contexte d'utilisation décrit par la localisation de l'utilisateur (plus spécifiquement, sa ville) et par la taille de l'écran du dispositif utilisé.

Dans ce qui suit, nous présentons l'ensemble des techniques d'adaptation qui est réalisable en utilisant ProbCWS. La représentation de ces techniques destinée à enrichir le modèle est décrite ensuite. Enfin, une évaluation de la mise en œuvre de l'adaptation par le biais des différents critères exposés lors de la synthèse de l'état de l'art (*cf.* section 6.4.3), termine cette section.

8.7.1 Mise en œuvre de l'adaptation au contexte d'utilisation

Par l'intermédiaire de ProbCWS, l'ensemble des formes d'adaptation fournies dans les travaux proposant de l'adaptation dans le domaine des services Web est rendu possible. Ces formes d'adaptation sont :

- l'adaptation du résultat du service,
- l'adaptation de la demande de services,
- l'adaptation de la composition.

Bien que le modèle ProbCWS soit dédié à la composition de services Web, les deux premières formes d'adaptation (du résultat et de la demande), propres aux services Web élémentaires, sont intégrées dans la mise en œuvre de la composition, à chaque étape de celle-ci (*i.e.* pour chaque service Web composant).

Les formes d'adaptation sont réalisées par le biais de techniques d'adaptation. Chaque technique d'adaptation est mise en œuvre de manière différente dans ProbCWS. C'est pourquoi nous avons choisi d'organiser cette partie portant sur la mise en œuvre de l'adaptation dans ProbCWS selon les différentes techniques d'adaptation : par déploiement de services d'adaptation, par compensation, par sélection de services Web, et par sélection de compositions.

8.7.1.1 Technique d'adaptation par déploiement de services d'adaptation

La technique d'adaptation par déploiement de services d'adaptation permet de mettre en œuvre la forme d'**adaptation du résultat du service**. Cette technique repose sur des services Web spécifiques mis à disposition des clients par les fournisseurs. Ces services Web sont appelés dans notre travail des services Web d'adaptation. Une définition plus détaillée est présente dans la section décrivant la mise en œuvre de l'adaptation au sein de WSR-Model (*cf.* section 7.4).

ProbCWS traite l'invocation des services Web d'adaptation comme n'importe quels autres services Web. L'invocation est réalisée dans l'instruction d'une tâche élémentaire (*cf.* section 8.5.3.1).

Nous déterminons une technique d'adaptation par déploiement de services d'adaptation particulière : la **technique d'adaptation par conversion**. Cette technique consiste à prendre comme paramètres d'entrée les sorties du service le précédant. La sortie de ce service Web d'adaptation est, soit l'entrée du service Web le succédant, soit la sortie de la tâche principale. Dans ProbCWS, cette technique d'adaptation est réalisée lors de l'invocation d'une tâche élémentaire, par l'appel d'un service Web d'adaptation spécifique. Cette invocation peut être définie par deux acteurs : soit par le client, soit par le moteur d'inférence.

Adaptation par déploiement de services d'adaptation réalisée par le client. Lors de la définition de la composition de services avec ProbCWS, le client peut faire appel dans la composition à un service Web d'adaptation. Ce service peut convertir des types de données afin que ces valeurs conviennent aux entrées des services suivants. Cet appel de service se fait comme tout autre appel de service dans la définition de la composition ProbCWS.

Adaptation par déploiement de services d'adaptation réalisée par le moteur d'inférence. Lors de l'exécution d'une composition, le moteur d'inférence peut mettre en œuvre d'autres techniques d'adaptation (telles que la compensation, la sélection de services, ou la sélection de compositions – *cf.* paragraphes suivants). Dans certains cas, les types d'entrée ou de sortie du nouveau service (ou de la nouvelle composition) appelé(e) ne conviennent pas par rapport à la composition initiale. Dans ce cas, le moteur d'inférence cherche dans la base de connaissances du domaine un service Web d'adaptation qui permet de convertir ces entrées/sorties.

Cette technique d'adaptation est illustrée par l'exemple d'utilisation de ProbCWS donné par la composition *myMeteo* (*cf.* section 8.5.5). Dans cet exemple, le client définit lors de la formalisation d'une tâche élémentaire, l'appel à un service Web d'adaptation de conversion qui prend en entrée un fichier XML et le convertit en format DIAL.

En utilisant l'arbre de résolution ProbCWS, la représentation graphique des techniques d'adaptation par déploiement de services d'adaptation est réalisée via l'appel d'une tâche élémentaire d'adaptation.

8.7.1.2 Technique d'adaptation par compensation

La technique d'adaptation par compensation de services Web permet de mettre en œuvre les formes d'**adaptation de la demande de services Web** et **de la composition**. Cette technique, réalisée lors de l'étape de surveillance de la composition, consiste à invoquer un autre service Web (ou une autre composition) lorsque l'appel du premier service Web échoue.

La Figure 8.22 illustre l'invocation d'un service choisi par le client dans la tâche élémentaire *geoIP*. Si le service échoue, la compensation est réalisée en trois étapes.

(1) Le moteur d'inférence recherche dans la base de connaissances du domaine la **catégorie de services** à laquelle le service appelé par la tâche *geoIP* appartient.

(2) Le moteur d'inférence recherche dans la base de connaissances des problèmes une **tâche élémentaire qui invoque un service de la même catégorie** que le service qui a échoué.

(3) Si **la signature de la nouvelle tâche est identique** à la tâche élémentaire qui faisait appel au premier service, alors le moteur d'inférence l'invoque (il s'agit de l'exemple illustré à gauche de la Figure 8.22 avec l'invocation de la tâche élémentaire *newET1*).

(3') Si **la signature de la nouvelle tâche est différente** de celle de la tâche *geoIP*, le moteur d'inférence réalise, en plus de la compensation, la technique d'adaptation par conversion, définie

dans le paragraphe précédent. Le moteur d'inférence invoque la tâche élémentaire *newET2* qui elle-même invoque deux autres tâches élémentaires : *CS1* qui fait appel à un service classique, qui permet la compensation de la tâche *geoIP* ; et *CS2* qui fait appel à un service d'adaptation qu met en œuvre la conversion de la sortie de la tâche *CS1* pour convenir à la sortie de la résolution du problème *ipToTown* (*cf.* Figure 8.22, à droite).

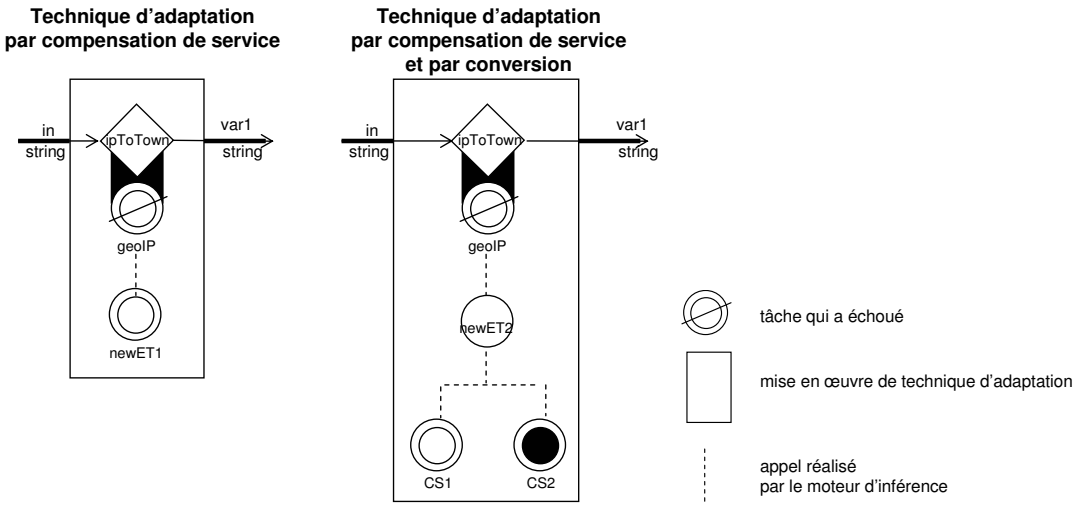


Figure 8.22 Illustration de la mise en œuvre de la technique d'adaptation par compensation à l'aide de l'arbre de résolution ProbCWS.

8.7.1.3 Technique d'adaptation par sélection de services Web

La technique d'adaptation par sélection de services Web permet de mettre en œuvre les formes d'**adaptation de la demande de services Web** et **de la composition**. Cette technique est réalisée lors de la définition de la composition de services Web avec ProbCWS et s'appuie sur deux étapes.

(1) Le client renseigne, lors de la définition de la composition, la catégorie de services Web qu'il souhaite invoquer dans l'instruction de la tâche élémentaire (*cf.* la définition des instructions d'une tâche élémentaire section 8.5.3.1).

(2) Lors de l'exécution de la composition, le moteur d'inférence recherche dans la base de connaissances du domaine l'ensemble des services Web appartenant à la catégorie de services indiquée par le client. La liste des services Web résultant de cette recherche est ordonnée selon un système de pondération[68] (*cf.* section 10.3.2.3). Le moteur d'inférence invoque le service Web retenu en premier dans la liste des résultats. Si ce service n'est pas disponible, le moteur d'inférence appelle le second et ainsi de suite.

La Figure 8.23 illustre l'utilisation de la technique d'adaptation par sélection de services Web. Au sein de la définition des instructions de la tâche *geoIP*, le client souhaite appeler un service de la catégorie *LocalizationByIP*. L'arbre de résolution ProbCWS représentant cette tâche montre que la tâche *geoIP* est résolue par le moteur d'inférence via une spécialisation. À la suite de l'étape de sélection, deux services Web sont trouvés : *CS1* et *CS2*. Ces deux services Web appartiennent à la catégorie *LocalizationByIP* et respectent les critères définis par le client (paramètre d'entrée un entier, paramètre de sortie une chaîne de caractères, et adapté au contexte d'utilisation composé de l'élément *localization*). Selon le mécanisme de pondération, le moteur d'inférence invoque l'un ou l'autre de ces services.

[68] Ce mécanisme n'étant pas en lien direct avec le modèle de composition ProbCWS, il n'est pas décrit ici.

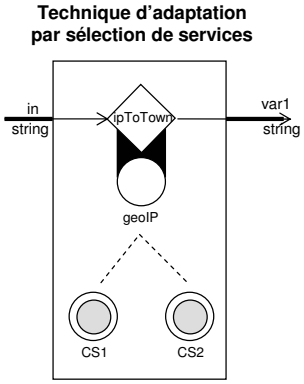


Figure 8.23 Illustration de la mise en œuvre de la technique d'adaptation par sélection de services Web via un arbre de résolution ProbCWS.

8.7.1.4 Technique d'adaptation par sélection de compositions

La technique d'adaptation par sélection de compositions permet de mettre en œuvre la forme d'**adaptation de la composition**. Elle est réalisée en deux étapes.

(1) Lors de la définition d'une composition, le client définit dans l'instruction de la tâche élémentaire l'appel d'une catégorie de compositions. Une catégorie de compositions est représentée dans la base de connaissances du domaine par la classe *CompositionCategory* du package *Composition* (*cf.* section 8.5.2).

(2) Lors de l'exécution de la composition, le moteur d'inférence cherche dans la base de connaissances du domaine les compositions appartenant à la catégorie de la composition. Le résultat de cette recherche est une liste de compositions triée selon un mécanisme de pondération[69] (*cf.* section 11.3.1.2). Le moteur fait appel à la première composition. Si cette dernière échoue, il fait appel à la composition suivante et ainsi de suite.

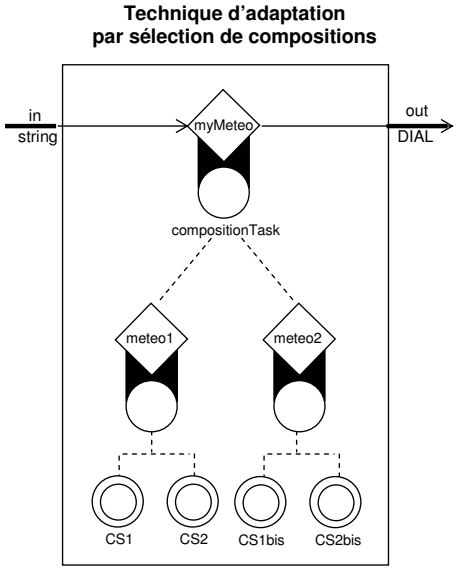


Figure 8.24 Illustration de la technique d'adaptation par sélection de compositions.

La Figure 8.24 illustre un exemple d'utilisation de la technique d'adaptation par sélection de compositions réalisée par le moteur d'inférence. Le client, lors de la définition de la tâche élémentaire

[69] Le mécanisme de pondération n'étant pas directement lié au modèle ProbCWS, il n'est pas décrit dans ce chapitre.

compositionTask, décrit dans les instructions de la tâche l'appel à la catégorie de compositions *meteo*. Cette catégorie est composée de deux compositions (*i.e.* deux tâches principales) : *meteo1* et *meteo2*.

8.7.2 Représentation de la mise en œuvre de l'adaptation au contexte d'utilisation dans le modèle ProbCWS

Le modèle de composition de services Web ProbCWS intègre l'ensemble des formes et des techniques d'adaptation que nous avons relevé dans le domaine de l'adaptation des services Web. Le package *Adaptation* enrichit le modèle ProbCWS et permet de représenter les différentes formes et techniques d'adaptation mises en jeu dans la définition d'une composition réalisée avec ce modèle (*cf.* Figure 8.25).

La classe *AdaptationApproach*. Cette classe représente les formes d'adaptation permises via ProbCWS. L'attribut *name* de cette classe peut prendre pour valeur : *resultAdaptation* (adaptation du résultat), *requestAdaptation* (adaptation de la demande de services), et *compositionAdaptation* (adaptation de la composition). Une forme d'adaptation peut être réalisée par une ou plusieurs techniques d'adaptation (association entre les classes *AdaptationApproach* et *AdaptationTechnique*).

La classe *AdaptationTechnique*. Cette classe représente l'ensemble des techniques d'adaptation réalisables avec ProbCWS. Une technique d'adaptation s'adapte à une description de contexte donnée (association entre les classes *AdaptationTechnique* et *ContextDescription* du package *Use Context*). Nous avons identifié trois grandes familles de techniques d'adaptation : adaptation par compensation (classe *Compensation*), adaptation par déploiement de services d'adaptation (classe *AdaptationServiceImplementation*), adaptation par sélection (classe *Selection*). Les deux dernières techniques sont spécialisables :

- *Adaptation par déploiement de services d'adaptation* : La technique d'adaptation par conversion (classe *Conversion*) hérite de la technique d'adaptation par déploiement de services d'adaptation puisqu'elle requiert la réalisation de tels services.
- *Adaptation par sélection* : La technique d'adaptation par sélection est spécialisable en la technique de sélection de services (classe *ServiceSelection*) et en la technique de sélection de compositions (classe *CompositionSelection*).

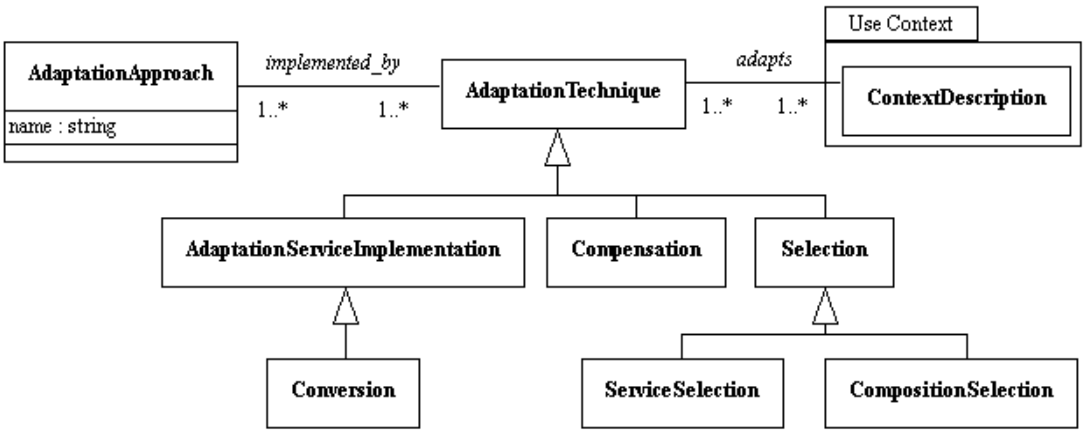


Figure 8.25 Diagramme de classes représentant le package *Adaptation* et le lien avec le package *Use Context*.

8.7.3 Évaluation de la mise en œuvre de l'adaptation avec ProbCWS

Le modèle de composition de services Web ProbCWS permet de mettre en œuvre l'ensemble des formes et des techniques d'adaptation issues du domaine des services Web (*cf.* Tableau 8.1).

	Formes d'adaptation		
	Adaptation du résultat	**Adaptation de la demande de services**	**Adaptation de la composition**
Techniques d'adaptation	- Conversion - Déploiement de service d'adaptation	- Sélection de SW - Compensation	- Sélection de SW - Compensation - Sélection de la composition

Tableau 8.1 Bilan des formes et des techniques d'adaptation au contexte d'utilisation mises en œuvre dans ProbCWS.

ProbCWS répond à l'ensemble de critères d'évaluation des techniques d'adaptation : l'adaptation minimale, l'adaptabilité, et l'adaptativité (*cf.* Figure 8.26).

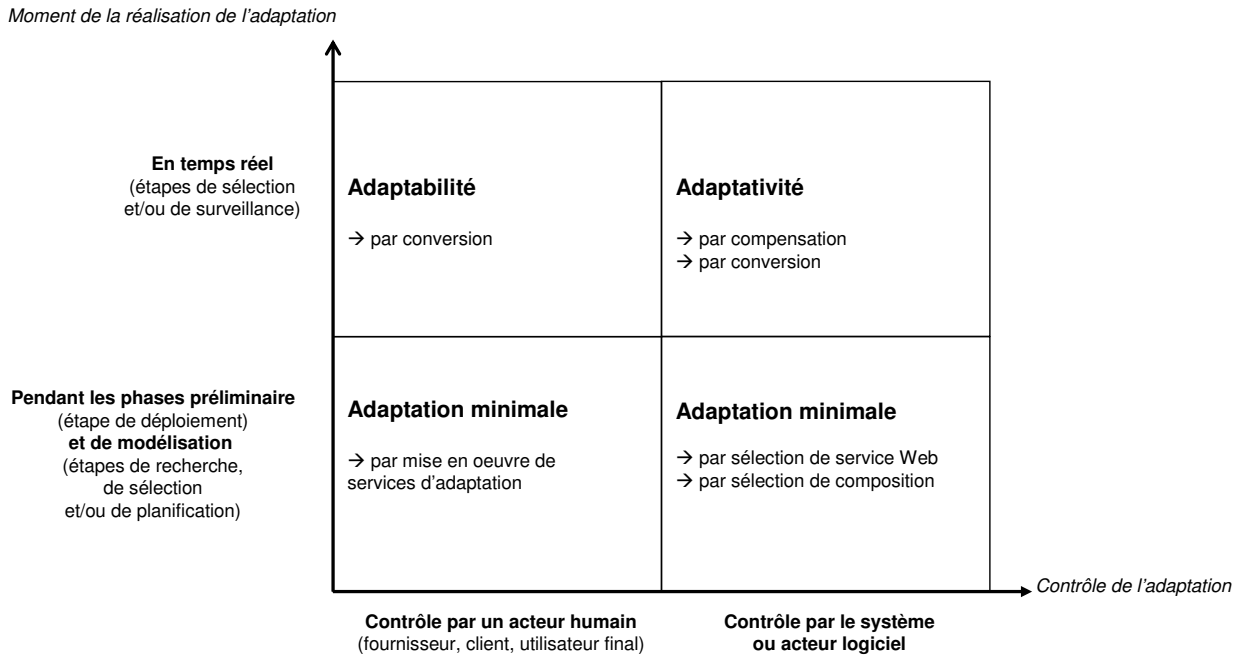


Figure 8.26 Critères d'évaluation de la mise en œuvre de l'adaptation avec ProbCWS.

L'adaptation minimale. ProbCWS respecte le critère d'évaluation d'adaptation minimale étant donné que ce modèle met en œuvre l'ensemble des formes d'adaptation (de la demande, du résultat, et de la composition) lors des phases préliminaires et de modélisation du cycle de vie des applications à base de services Web. Ces formes d'adaptation sont réalisées soit par :

- des acteurs humains (tels que le fournisseur) pour la technique de **déploiement de service d'adaptation**.
- le système ou des acteurs logiciels (tels que d'autres services Web) pour la **sélection de services Web et de compositions**.

L'adaptabilité. ProbCWS respecte le critère d'évaluation de l'adaptabilité du fait qu'il permet la mise en œuvre de la technique d'adaptation par **conversion** qui s'effectue lors de l'étape de sélection. Cette technique consiste à convertir le résultat d'un service Web afin de convenir à l'entrée du service Web le succédant en invoquant des services Web d'adaptation. Cette

technique est formalisée lors de la définition de la composition par le client et permet l'adaptation du résultat du service afin qu'il convienne au résultat désiré de la composition.

L'adaptativité. ProbCWS respecte le critère d'évaluation de l'adaptativité du fait qu'il permet la mise en œuvre des techniques d'adaptation par compensation et par conversion qui s'effectuent lors de l'étape de surveillance de la composition.

- Par le biais de la **compensation**, si un service Web échoue, il est remplacé par un autre service de même catégorie. La compensation est définie lors de la définition de la composition et est réalisée par le moteur d'inférence (système mettant en œuvre ProbCWS).
- La technique d'adaptation de **conversion** permet de convertir le résultat d'un service afin qu'il convienne, soit à l'entrée du service suivant, soit au résultat final de la composition.

8.8 Conclusion

Ce chapitre a été consacré à la présentation du modèle de composition de services Web que nous proposons dans le cadre de notre travail. Ce modèle a pour originalité de se baser sur des concepts issus du domaine de l'intelligence artificielle et de la résolution de problèmes en vue de définir la demande du client en terme de composition de services Web et de mettre en œuvre d'adaptation au contexte d'utilisation.

Dans la section 8.1, nous avons défini un ensemble de besoins auxquels le modèle de composition de services Web ProbCWS répond. Nous proposons d'en faire un bilan :

Utilisation d'un outil formel. Nous utilisons les méthodes de résolution de problèmes par modèle de tâches comme base de notre modèle du fait que ces méthodes permettent de :

- **conceptualiser la requête du client comme un problème à résoudre** et sa décomposition en tâches élémentaires comme des services Web.
- **séparer les connaissances des problèmes, les inférences, et les connaissances du domaine** en vue de réutiliser les services Web.

Intégration de la sémantique. L'architecture de base des applications utilisant des méthodes de résolutions de problèmes est composée d'une base de connaissances des problèmes, d'un moteur d'inférence, et d'une base de connaissances du domaine. L'ensemble de ces trois entités permet de manipuler des objets à un niveau sémantique.

Intégration de l'adaptation au contexte d'utilisation. L'utilisation d'une méthode de résolution de problèmes permet d'ajouter des dimensions à prendre en compte lors de la résolution de ces derniers. Nous avons choisi d'intégrer le concept d'adaptation au contexte d'utilisation au problème à résoudre (autrement dit à la requête de composition de services Web).

La définition d'une composition de services Web à l'aide de ProbCWS repose sur la description d'un ensemble de triplets : tâche principale, tâche élémentaire, et stratégie.

La tâche principale. Cette tâche représente le problème à résoudre demandé par le client.

La tâche élémentaire. Les tâches élémentaires représentent l'exécution proprement dite de la résolution du problème. Elles peuvent appeler des tâches, des services concrets, des catégories de services, ou des catégories de composition.

La stratégie. La stratégie permet d'associer une tâche élémentaire à une tâche principale.

En utilisant le modèle de composition de services Web ProbCWS, nous mettons en œuvre l'adaptation au contexte d'utilisation à travers deux points : l'enrichissement de la requête du client et la mise en œuvre de formes d'adaptation par le biais de techniques d'adaptation.

L'enrichissement de la requête du client. ProbCWS permet aux clients d'intégrer à leur requête de composition le contexte d'utilisation auquel il souhaite que le résultat soit adapté.

La mise en œuvre de formes d'adaptation. ProbCWS permet de mettre en œuvre trois formes d'adaptation au contexte d'utilisation : l'adaptation du résultat, l'adaptation de la demande de services Web, et l'adaptation de la composition.

Le chapitre qui suit montre comment sont opérationnalisés les deux modèles qui viennent d'être décrits (le modèle de représentation de services – WSR-Model, et le modèle de composition – ProbCWS).

9 OPÉRATIONNALISATION DES MODÈLES

Nous venons d'étudier les deux modèles issus de notre proposition concernant la représentation des services Web (WSR-Model) et la composition de services Web (ProbCWS).

WSR-Model est un modèle qui enrichit la description des services Web. Cette représentation de services facilite les étapes de recherche et de sélection pour les clients qui font suites à l'étape de publication réalisée par les fournisseurs.

ProbCWS est un modèle de composition de services Web qui repose sur les méthodes de résolution de problèmes par modèle de tâches. Ce modèle permet de mettre en œuvre les étapes de planification et de surveillance réalisées par les clients, en y intégrant de l'adaptation au contexte d'utilisation.

La définition de ces deux modèles répose sur une formalisation à base d'objets. Ce chapitre met en relief nos choix technologiques en vue d'implémenter ces modèles. Il est composé de deux sections décrivant nos choix d'opérationnalisation concernant, respectivement, le modèle de représentation de services Web (WSR-Model), et le modèle de composition de services Web (ProbCWS).

9.1 Opérationnalisation de WSR-Model

Cette section étudie les choix technologiques concernant l'opérationnalisation de WSR-Model. Nous définissons dans un premier temps dans quel contexte ce modèle peut être utilisé, pour, par la suite, choisir l'outil adéquat avec lequel le modèle va être opérationnalisé. Une fois cet outil décrit, nous mettons en relief les mises en correspondance nécessaires cette mise en œuvre des modèles.

9.1.1 Utilisation de WSR-Model

Le modèle de représentation de services Web a pour objectif d'enrichir la description des services afin de faciliter les étapes de recherche et de sélection pour les clients. Ces étapes sont possibles dans la mesure où les fournisseurs publient leurs services selon le modèle WSR-Model. Ces trois étapes (publication, recherche, et sélection) se font via des registres (*cf.* chapitre 3).

Dans le domaine des services Web, un registre est vu comme une base de descriptions de services Web. D'après [Curbera *et al.*, 2002], deux notions définissent un registre :

- un registre **stocke la représentation** des services publiés ;
- un registre est **un moyen d'interroger** et de mettre à jour les informations qui décrivent les services.

La fonctionnalité d'un registre est donc double et concerne deux types d'utilisateur : les **fournisseurs** qui publient leurs services, et les **clients** qui interrogent les bases de représentations de services.

D'après la définition d'un registre, l'architecture d'un tel outil doit reposer sur deux modules : le gestionnaire de connaissances et le gestionnaire de fonctionnalités (*cf.* Figure 9.1).

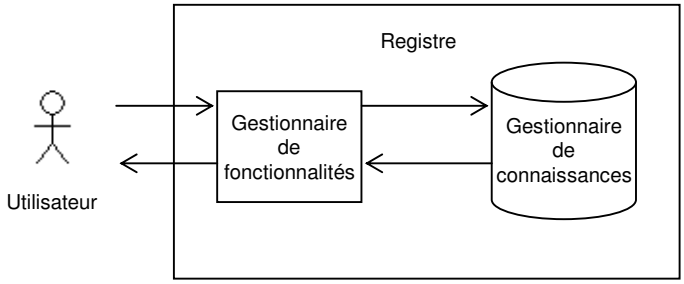


Figure 9.1 Architecture simplifiée d'un registre.

Le gestionnaire de connaissances a pour objectif de stocker la connaissance du domaine. Dans notre contexte de travail, cette connaissance est la représentation des services publiés. WSR-Model est utilisé comme base de l'organisation des connaissances dans ce module.

Le gestionnaire de fonctionnalités met en œuvre les méthodes fournies aux utilisateurs du registre (fournisseurs et clients). Dans le cadre d'un registre de services, ces méthodes sont la publication et la recherche de services. Ces fonctionnalités permettent d'interroger et de mettre à jour les représentations des services stockées dans le gestionnaire de connaissances.

La section qui suit décrit l'outil avec lequel nous avons choisi d'implémenter le registre et ainsi d'opérationnaliser WSR-Model.

9.1.2 Choix de l'outil d'opérationnalisation

Nous décrivons ici une argumentation concernant le choix de l'outil d'opérationnalisation ainsi que les principes de base de cet outil.

9.1.2.1 Arguments concernant le choix d'AROM

Afin d'implémenter les deux parties composant un registre (gestionnaire de fonctionnalités et de connaissances), nous avons choisi d'utiliser le Système de Représentation de Connaissances par Objets (SRCO) nommé AROM (Associer Relations et Objets pour Modéliser)[70] [Page *et al.*, 2001].

AROM se présente sous la forme d'un ensemble d'interfaces de programmation (API) Java. Ce système reprend les principes classiques de la représentation de connaissances par objets, tels que la distinction entre classes et instances, la spécialisation de classes, et la présence de facettes de typage. Le système AROM se démarque des autres SRCO notamment par la représentation explicite des associations entre les objets. La majorité des SRCO ne disposent typiquement que du concept de classe. L'utilisation d'AROM apporte des avantages certains lors de l'implémentation des gestionnaires de connaissances et de fonctionnalités.

[70] http://www.inrialpes.fr/romans/pub/arom

Le gestionnaire de connaissances. Le système de représentation de connaissances AROM permet la gestion des bases de connaissances du domaine. Le choix d'AROM se base sur le fait que le modèle de représentation de services Web étant formalisé par le biais d'une représentation par objets, l'utilisation d'un SRCO est directe puisqu'il propose des concepts et une notation qui sont propres à ce type de représentation de connaissances.

Le gestionnaire de fonctionnalités. AROM possède des avantages en vue d'implémenter les deux fonctionnalités de WSR : la publication et la recherche de services Web.

- **La publication**. Le fait que AROM soit un SRCO facilite la publication des services basée sur le modèle de représentation de services Web lui-même orienté objet. De plus, nous avons aussi proposé une ontologie, nommée OntoWS (*cf.* annexe 1), basée sur WSR-Model. Dans le cas où la publication se fait par l'intermédiaire de OntoWS, la transformation en AROM est établie par l'intermédiaire d'un travail préalable [Miron *et al.*, 2007].
- **La recherche**. Afin de permettre les inférences d'une valeur d'une variable ou d'interroger les bases de connaissances, AROM propose un langage (Langage de Modélisation Algébrique – LMA). Ce langage est utile à l'implémentation de la fonctionnalité de recherche sous-jacente aux registres puisqu'il permet d'interroger les bases de connaissances.

L'utilisation d'AROM apporte aussi deux autres avantages : l'intégration de la sémantique dans les registres de services Web et la possibilité d'étendre WSR-Model.

L'intégration de la sémantique. En utilisant AROM nous créons des ontologies issues du modèle de représentation de services Web. Le résultat de WSR-Model et de son implémentation dans un registre de services à l'aide d'AROM, répond à la définition d'une ontologie donnée dans l'état de l'art (celle de [Gruber, 1993]). La conceptualisation est apportée par le modèle. La spécification explicite est mise en œuvre lors de l'implémentation en AROM du modèle. Par conséquent, le modèle de représentation ainsi que sa mise en œuvre par l'intermédiaire d'AROM garantissent que les registres sous-jacents à l'opérationnalisation de WSR-Model intègrent la sémantique lors de la publication et de la recherche de services.

L'extension du modèle de représentation. La mise en œuvre du modèle de représentation de services Web par l'intermédiaire du système AROM permet de le rendre extensible. Nous pourrons par la suite imaginer de nouveaux concepts à prendre en compte dans le modèle tels que la modélisation de données spatio-temporelles. Nous faisons ici plus particulièrement référence aux travaux de [Moisuc, 2007] proposant une représentation spatio-temporelle à l'aide d'AROM.

9.1.2.2 Principes de base d'AROM

Nous n'évoquons ici que les concepts de base d'AROM dont nous avons besoin afin d'opérationnaliser WSR-Model[71], c'est-à-dire les concepts sous-jacents à la modélisation d'une base de connaissances objets (les classes, les objets, les associations, les rôles, et les tuples).

Classe et objet. Une classe décrit un ensemble d'objets ayant des propriétés et des contraintes communes. Ces propriétés qui caractérisent la classe sont appelées des variables [Page *et al.,* 2000]. Dans l'exemple illustré dans la Figure 9.2[72], la base de connaissances possède quatre

[71] La description de l'ensemble des spécificités d'AROM est disponible à l'adresse suivante : http://www.inrialpes.fr/romans/pub/arom/

[72] Cet exemple est un extrait de celui donné avec l'IME (*Integrated Modeling Environment*) d'AROM.

classes (*Enseignant*, *Permanent*, *Temporaire*, et *Cours* – lignes 3 à 32). Les classes peuvent être organisées par la relation de spécialisation qui, en AROM, supporte seulement un héritage simple [Bruley *et al.,* 2003]. Dans l'exemple, La classe *Enseignant* spécialise les classes *Permanent* et *Temporaire* (lignes 15 et 20).

Association, rôle, et tuple. Une *association* permet de lier au minimum deux classes. Dans l'exemple donné, il existe une association nommée *Enseigne* (lignes 35 à 49). Les liens entre l'association et les classes sont établis par des *rôles*. Dans l'exemple, l'association *Enseigne* est liée aux classes *Enseignant* et *Cours* respectivement par les rôles *enseignant* et *cours* (lignes 36 à 46). La multiplicité de chaque rôle est décrite par une facette pour laquelle est affectée une valeur minimum et une valeur maximum (lignes 39 à 41 et 44 à 46). L'instance d'une association est appelée un *tuple*.

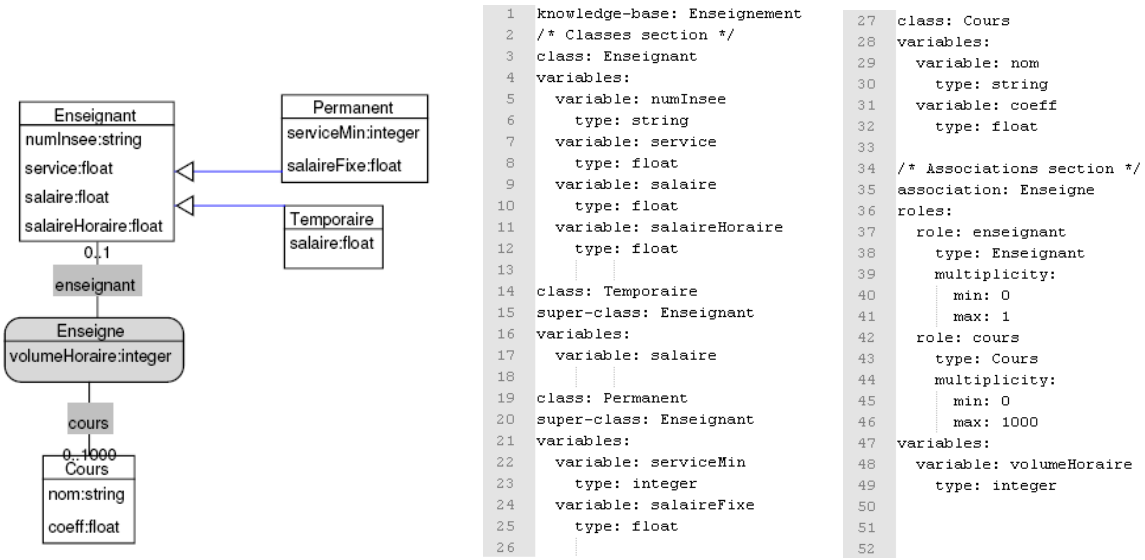


Figure 9.2 Principes élémentaires des bases de connaissances AROM représentés, à gauche, graphiquement, et à droite, textuellement.

9.1.3 Mises en correspondance nécessaires à l'opérationnalisation de WSR-Model avec AROM

Le Tableau 9.1 résume les mises en correspondance entre le modèle WSR-Model et son opérationnalisation avec AROM. Les mises en correspondance sont directes étant donné que WSR-Model repose sur un formalisme orienté objet à l'instar d'AROM. La seule spécificité concerne les associations.

WSR-Model	Opérationnalisation avec AROM
5 packages	La base de connaissances
22 classes	22 classes
4 relations de spécialisation	4 relations de spécialisation
37 associations	37 associations 74 rôles

Tableau 9.1 Tableau récapitulant les mises en correspondance nécessaires pour l'opérationnalisation de WSR-Model à l'aide du système AROM.

Les associations du système AROM possèdent des rôles afin de les lier avec les classes (*cf.* section 9.1.2.2). L'opérationnalisation de WSR-Model à l'aide d'AROM contient autant d'associations que WSR-Model et deux fois plus de rôles que d'associations, étant donné que WSR-Model ne possède que des associations binaires. Concernant le nom des associations lors de l'opérationnalisation, nous ne pouvons garder les mêmes noms que ceux utilisés dans WSR-Model. Dans WSR-Model plusieurs associations possèdent le même nom, or AROM n'autorise pas deux associations du même nom. Nous avons choisi les conventions de nommage suivantes pour les associations et les rôles :

Nom des associations. Les associations prennent pour nom les noms des classes liées séparées d'un 2. Par exemple, l'association nommée *Method2ConcreteService* lie les classes *Method* et *ConcreteService*.

Nom des rôles. Les rôles prennent le même nom que la classe qu'il lie à l'association. Par exemple, le rôle qui lie la classe *Method* à l'association *Method2ConcreteService* a pour nom *method*.

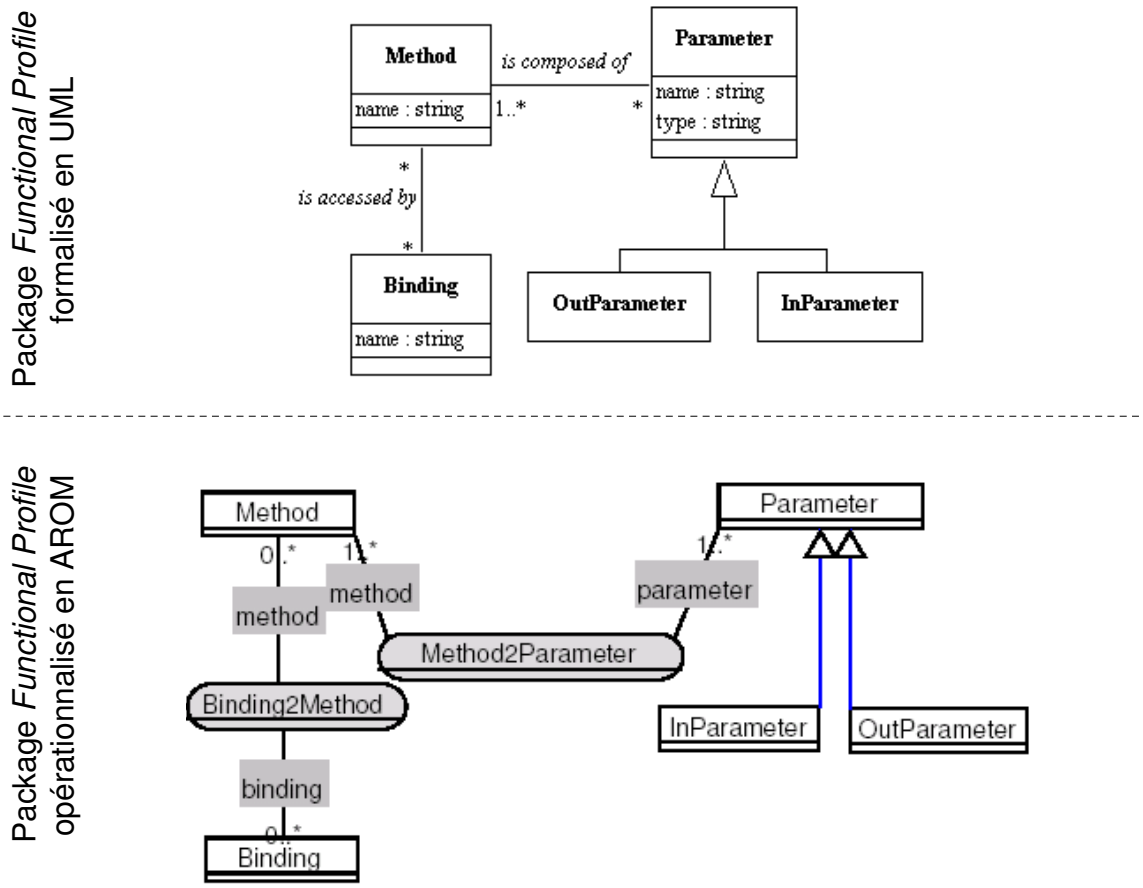


Figure 9.3 Package *Functional Profile* de WSR-Model (en haut) et son opérationnalisation en AROM (en bas).

Nous avons choisi d'illustrer l'opérationnalisation de WSR-Model avec la mise en correspondance du package *Functional Profile* de WSR-Model en AROM. La Figure 9.3 montre en haut la modélisation UML du package, et en bas, la modélisation en AROM du même package via l'utilisation de l'IME (*Integrated Modeling Environment*) d'AROM. Nous résumons l'opérationnalisation du package via la description de chacun des concepts mis en jeu dans la modélisation : les classes, les relations de spécialisation, et les associations.

Les classes. L'opérationnalisation du package contient le même nombre de classes et ces dernières ont gardé le même nom.

Les relations de spécialisation. La seule relation de spécialisation du package *Functional Profile* est celle entre les classes *Parameter* et *InParameter* et *OutParameter*. L'opérationnalisation en AROM restitue ce type de relation de la même manière.

Les associations. Le package et son opérationnalisation contiennent deux associations. Les changements à noter sont :

- *le changement de nom des associations* selon la convention décrite plus haut. Par exemple l'association nommée *is composed of* entre les classes *Method* et *Parameter* a pour nom, au sein de l'opérationnalisation avec AROM, *Method2Parameter*.
- *l'ajout de rôles*. Afin de lier les associations avec les classes, AROM ajoute la notion de rôle. Le nom des rôles respecte la convention décrite plus haut. Les multiplicités restent les mêmes.

9.2 Opérationnalisation de ProbCWS

Le second modèle que nous proposons est ProbCWS. Ce modèle propose un moyen de définir la composition de services Web en y intégrant la mise en œuvre de l'adaptation au contexte d'utilisation. Dans un premier temps, l'utilisation de ProbCWS est présentée. Nous décrivons dans un second temps AROMTasks, le langage que nous avons choisi afin d'opérationnaliser ce modèle, puis les mises en correspondance entre ProbCWS et ce langage. Afin de mettre en œuvre ProbCWS via AROMTasks, ce dernier doit être étendu. Le dernier paragraphe décrit les extensions nécessaires.

9.2.1 Utilisation de ProbCWS

ProbCWS a été défini pour les applications nécessitant une définition et l'exécution de compositions de services Web. Étant donné que ce modèle repose sur une méthode de résolution de problèmes, les applications l'utilisant doivent respecter une architecture pré-déterminée. Ce type d'architecture a déjà été étudié dans le chapitre portant sur ProbCWS (*cf.* chapitre 8). Nous rappelons brièvement ici les modules la composant (*cf.* Figure 9.4).

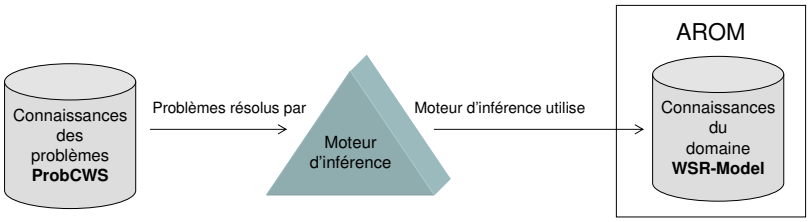


Figure 9.4 Rappel de l'architecture préalable à la mise en œuvre de ProbCWS.

La base de connaissances des problèmes. Cette base contient les définitions des problèmes et leur(s) résolution(s). Dans le cadre de notre travail, ces définitions sont exprimées en ProbCWS.

Le moteur d'inférence. Le moteur d'inférence réalise la résolution à partir des connaissances de problèmes et du domaine.

La base de connaissances du domaine. La base de connaissances du domaine stocke et organise les connaissances nécessaires au moteur d'inférence en vue de résoudre les problèmes. Nous avons choisi, afin d'opérationnaliser le modèle WSR-Model (*i.e.* la base de connaissances du domaine), le système de représentation de connaissances par objets AROM.

9.2.2 Choix de l'outil d'opérationnalisation

ProbCWS repose sur la résolution de problèmes par modèle de tâches. Afin d'opérationnaliser ce modèle et ainsi exécuter cette forme de résolution de problèmes, nous avons choisi AROMTasks [Genoud *et al.*, 2005], langage associé au système AROM. Ce langage a été défini afin d'exploiter les connaissances du domaine à partir des connaissances des problèmes.

Afin de décrire les principes de base de l'utilisation d'AROMTasks, nous décrivons brièvement l'environnement intégré d'AROM. Deux Bases de Connaissances (BC) sont identifiées dans AROM (*cf.* Figure 9.5) : la BC méthodologique et la BC du domaine. La BC méthodologique exploite les connaissances stockées dans la BC du domaine, et utilise ces connaissances en vue de paramétrer les méthodes de raisonnement (*i.e.* résoudre les problèmes). La BC du domaine organise en connaissances les données transmises dans des hiérarchies de classes et d'associations et stocke ces connaissances (ceci a été traité par l'opérationnalisation de WSR-Model par AROM – *cf.* section 9.1).

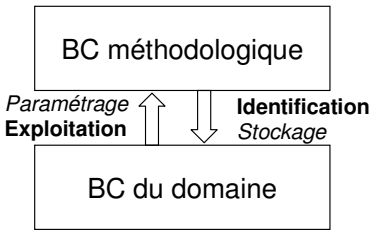


Figure 9.5 Environnement intégré d'AROM, d'après [Chabalier, 2004].

Dans notre travail, la base de connaissances du domaine permet de stocker la connaissance issue du modèle de représentation de services Web – WSR-Model (*cf.* chapitre 7). La base de connaissances méthodologique permet de modéliser les problèmes afin de mettre en œuvre, par la suite, leur résolution.

La résolution de problèmes dans AROMTasks repose sur trois concepts : le problème lui-même, la méthode, et la stratégie.

Le problème. La syntaxe d'un problème en AROMTasks est représentée par la Figure 9.6. Un problème est une liste d'entrées (élément `inputs` – lignes 4 et 5) et de sorties (élément `outputs` – lignes 6 et 7) typées.

```
1  problem ExempleSyntaxeProblème
2     documentation "ceci est la documentation du problème"
3
4     inputs
5       nomArgumentEntree typeArgumentEntree
6     outputs
7       nomArgumentSortie typeArgumentSortie
```

Figure 9.6 Syntaxe d'un problème dans AROMTasks.

La méthode. Les méthodes de résolution sont des fonctions[73] dont les arguments sont les paramètres des problèmes (éléments `inputs` et `outputs`). Le corps d'une méthode (élément `body` – lignes 9 à 11) appartient à la définition d'une méthode dans AROMTasks et peut être :

- un algorithme,
- un appel à une autre méthode,
- un appel à une résolution de problèmes.

[73] Dans le cas particulier d'AROMTasks, les fonctions sont représentées par des scripts inspirés de Java.

La syntaxe d'une méthode en AROMTasks est représentée par la Figure 9.7.

```
1   method exempleSyntaxeMéthode
2      documentation "ceci est la documentation du problème"
3
4      inputs
5        nomArgumentEntree typeArgumentEntree
6      outputs
7        nomArgumentSortie typeArgumentSortie
8
9      body
10      //corps de la méthode à insérer
11     bodyend
```

Figure 9.7 Syntaxe d'une méthode dans AROMTasks.

La stratégie. La stratégie permet d'attribuer une méthode (ou un ensemble de méthodes) à un problème. Une stratégie est liée à un problème à partir du moment où elle porte le même nom. Les méthodes qui peuvent résoudre le problème sont définies par les éléments `solver` (*cf.* lignes 4 et 5). Les instructions d'attribution d'une méthode ou d'une autre pour résoudre le problème sont décrites dans l'élément `criteria`. Dans AROMTasks, les instructions contenues dans l'élément `criteria` sont décrites par des fonctions Java qui renvoient le nom de la méthode à exécuter. La syntaxe d'une stratégie en AROMTasks est représentée par la Figure 9.8.

```
1   strategy exempleSyntaxeStratégie
2      documentation "ceci est la documentation de la strategie"
3
4      solver exempleSyntaxeMéthode
5      solver exempleSyntaxeMéthode1
6
7      criteria
8      //instruction à insérer
9      criteriaend
```

Figure 9.8 Syntaxe d'une stratégie dans AROMTasks.

9.2.3 Mises en correspondance des concepts de ProbCWS et ceux d'AROMTasks

La Figure 9.9 illustre les relations qui existent entre les concepts de base d'AROMTasks (problème, stratégie, et méthode) et ceux de ProbCWS (tâche principale, stratégie, et tâche élémentaire – *cf.* chapitre 8).

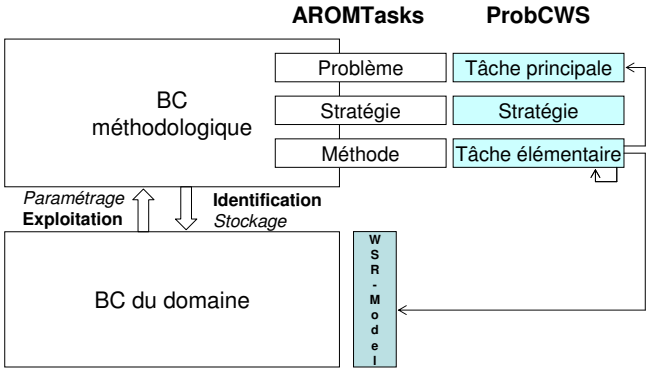


Figure 9.9 Illustration de la mise en correspondance des principes de base d'AROMTasks et de ProbCWS.

Étant donné que ProbCWS et AROMTasks sont tous les deux basés sur la méthode de résolution de problèmes à base de modèle de tâches, la mise en correspondance est directe :

- Le **problème** à résoudre d'AROMTasks correspond à la **tâche principale** de ProbCWS.
- AROMTasks et ProbCWS utilisent tous les deux une **stratégie** pour associer le problème à sa résolution.
- La **méthode** de résolution d'AROMTasks correspond à une **tâche élémentaire** de ProbCWS.

9.2.4 Proposition d'une extension d'AROMTasks : AROMTasks-CWS

La particularité de ProbCWS est qu'il intègre à la définition et à l'exécution de la composition de services Web (*i.e.* la résolution de problèmes) la mise en œuvre de l'adaptation au contexte d'utilisation. Afin d'atteindre cet objectif, nous proposons une version étendue d'AROMTasks (AROMTasks-CWS – *AROMTasks-Composition of Web Services*). Cette mise en œuvre repose principalement sur l'enrichissement de la définition d'une tâche et sur l'ajout, dans les instructions des stratégies et des tâches élémentaires, d'instructions d'appel spécifiques (de services Web, de catégories de services Web, et de compositions de services Web).

Nous définissons ici l'intégration des ces particularités à l'extension du langage AROMTasks-CWS.

9.2.4.1 Mécanismes d'enrichissement de la définition d'un problème et d'une méthode

L'extension AROMTasks-CWS permet d'intégrer aux définitions d'un problème et d'une méthode, la possibilité de **spécifier le contexte d'utilisation** pour lequel le client souhaite une adaptation et **le domaine d'application** auquel il souhaite que la résolution de problèmes appartienne. Ceci correspond en ProbCWS, à la définition des signatures de tâches (*cf.* section 8.5.1.1).

Le mécanisme d'intégration du contexte d'utilisation et du domaine d'application consiste à ajouter deux éléments à la définition d'un problème et d'une méthode. Ces deux éléments (`useContext` et `applicationDomain`) sont traités au même niveau que les éléments représentant les entrées (`inputs`) et les sorties (`outputs`) du problème ou de la méthode[74]. Un exemple de définition de méthode avec AROMTasks-CWS est donné dans la Figure 9.10. La signature de la méthode `myAdaptedTask` est composée de quatre éléments : les entrées (ligne 2), les sorties (ligne 3), le contexte d'utilisation pour lequel la tâche est adaptée (ligne 4), et le domaine d'application auquel la tâche appartient (ligne 5).

```
1  method myAdaptedTask
2      inputs              in      integer
3      outputs             out     integer
4      useContext          time    hour
5      applicationDomain   domain  e-business
6      body
7          ...
8      endbody
```

Figure 9.10 Illustration du mécanisme d'intégration de la spécification du contexte d'utilisation et du domaine d'application aux méthodes AROMTasks-CWS.

74 Les grammaires AROMTasks-CWS des problèmes et des méthodes sont disponibles à l'adresse suivante : http://www-lsr.imag.fr/users/Celine/Lopez-Velasco/AROMTasks-CWS/grammaire

Cette particularité est traduite dans AROMTasks par l'ajout de nouvelles variables dans le corps de la méthode. Nous avons créé **deux classes** (UC et AD) représentant respectivement le contexte d'utilisation et le domaine d'application. L'intégration de ces concepts au sein de AROMTasks consiste à instancier de nouveaux objets issus de ces classes. L'exemple qui suit (*cf.* Figure 9.11) illustre le mécanisme d'intégration de la définition du contexte d'utilisation (lignes 5 et 6) et du domaine d'application (lignes 7 et 8) dans AROMTasks pour la tâche `myAdaptedTask`.

L'association d'un contexte d'utilisation à la tâche *myAdaptedTask* se déroule en deux temps : tout d'abord par la création d'un objet, instance de la classe UC (ligne 5) ; puis par l'instanciation de cet objet par les éléments du contexte. Dans notre exemple, il s'agit des éléments temps et heure (ligne 6).

L'association d'un domaine d'application à la tâche *myAdaptedTask* est aussi constituée de deux étapes : (1) la création d'un objet, instance de la classe AD (ligne 7) ; (2) l'instanciation de cet objet par la valeur du domaine d'application (ligne 8). Dans notre exemple, la tâche *myAdaptedTask* appartient au domaine d'application du *e-business*.

```
1   method myAdaptedTask
2       inputs  in  integer
3       outputs out integer
4       body
5           UC uc = new UC();
6           uc.add("time","hour");
7           AD ad = new AD();
8           ad.add("e-business");
9           ...
10      endbody
```

Figure 9.11 Illustration de l'intégration des spécifications AROMTasks-CWS dans AROMTasks.

9.2.4.2 Mécanismes d'invocation

Dans ProbCWS nous avons défini qu'une tâche élémentaire peut contenir un ensemble d'instructions par l'intermédiaire duquel les méthodes de résolution du problème sont connues (*cf.* section 8.5.3.1). Les instructions d'affectation et de contrôle (condition et itération) sont permises dans AROMTasks à l'aide de *BeanShell*[75]. *BeanShell* est un interpréteur Java qui permet le support d'un langage de script afin d'exécuter du code Java sans avoir besoin de créer une classe complète. Il est utilisé dans AROMTasks pour exprimer dans une stratégie l'élément `criteria` et dans une méthode l'élément `body`.

Dans ProbCWS, nous avons intégré trois types d'instructions d'invocation dans une tâche élémentaire : une instruction d'invocation d'un **service Web simple**, une instruction d'invocation d'une **catégorie de services Web** et une instruction d'invocation d'une **catégorie de compositions de services Web**. L'intégration de ces types d'instructions dans AROMTasks-CWS est étudiée dans ce qui suit.

L'ensemble des spécifications de ces invocations est intégré dans le corps de la méthode[76]. La Figure 9.12 illustre un exemple de chaque type d'appel. Dans cet exemple, nous appelons dans l'ordre un service (`myWS`), une catégorie de services (`Localisation`), et une catégorie de compositions (`Weather`). Quatre variables locales sont définies (lignes 7 à 10) afin de permettre les compositions entre les entrées/sorties des appels. À la variable `outWS` est affectée le résultat de l'appel du service `myWS` (ligne 12). À la variable `outWSCat` est affectée le résultat de l'appel de la catégorie de services

[75] http://www.beanshell.org

[76] La grammaire des spécifications des différents types d'appel est disponible à l'adresse suivante : http://www-lsr.imag.fr/users/Les.Personnes/Celine.Lopez-Velasco/AROMTasks-CWS/grammaire/Method

`Localisation` (ligne 13). À la sortie de la méthode (`out`) est affectée le résultat de l'appel de la catégorie de compositions `Weather` (ligne 14).

```
1   method myAdaptedTask
2       inputs              in      integer
3       outputs             out     integer
4       useContext          time    hour
5       applicationDomain   domain  e-business
6       body
7           outWS       string
8           inWSCat     string
9           outWSCat    integer
10          inWSComp    integer
11
12          outWS       = call myWS(in)
13          outWSCat    = call fromCategory Localisation(inWSCat)
14          out         = call fromCWSCategory Weather(inWSComp)
15      endbody
```

Figure 9.12 Illustration des mécanismes d'invocation dans AROMTasks-CWS.

Ces trois types d'invocation doivent être traduites dans la syntaxe d'AROMTasks. Pour cela nous avons réalisé **trois classes** :

- *CallWS*, pour l'appel d'un service Web simple.
- *CallWSCategory*, pour l'appel d'une catégorie de services Web.
- *CallWSCompCategory*, pour l'appel d'une catégorie de compositions de services Web.

Chacune de ces classes possède une méthode d'invocation, nommée *invoke*, qui prend en paramètre :

- le nom du concept à invoquer, c'est-à-dire un service Web simple pour la classe *CallWS*, une catégorie de services Web pour la classe *CallWSCategory*, ou une catégorie de compositions de services Web pour la classe *CallWSCompoCategory* ;
- un tableau représentant les valeurs des paramètres d'entrée et le nombre de paramètres de sortie ;
- le contexte d'utilisation ;
- le domaine d'application.

```
1   method myAdaptedTask
2       inputs  in  integer
3       outputs out integer
4       ...
5       body
6           ...
7           ObjectValue inWS = new Objectvalue (in);
8           ObjectValue outWS = new ObjectValue();
9           Object[] params = new Object[]{inWS,outWS}
10
11          CallWS.invoke("myWS",params, uc, ad);
12          ...
13      endbody
```

Figure 9.13 Illustration de l'intégration des spécifications d'invocation dans AROMTasks.

L'exemple donné illustre l'invocation du service Web `myWSTask` (*cf.* Figure 9.13) à l'aide d'AROMTasks. L'appel de la méthode `invoke` (ligne 11) prend en paramètre le nom du service, le

tableau de paramètres (`params`) ainsi que le contexte d'utilisation (`uc`) et le domaine d'application (`ad`). La classe `ObjectValue` (lignes 7 et 8) est une classe auxiliaire qui permet d'englober les valeurs effectives des paramètres d'entrée et de sortie. Chaque paramètre est automatiquement associé à une instance de la classe `ObjetValue` et est transmis à la méthode d'invocation (lignes 9 et 11).

9.3 Conclusion

Nous avons décrit dans les deux chapitres précédents les deux modèles que nous proposons afin de faciliter les tâches des acteurs d'un cycle de vie d'une application à base de services. WSR-Model, modèle de représentation de services Web, enrichit l'étape de recherche des clients afin qu'il trouve les services Web les plus adéquats à leurs besoins. ProbCWS, modèle de composition de services Web, permet de définir les compositions de services Web et les requêtes de composition de services Web en y intégrant le concept d'adaptation au contexte d'utilisation.

L'opérationnalisation de WSR-Model repose sur la mise en œuvre de deux modules : un gestionnaires de connaissances (pour le stockage des représentations de services) et un gestionnaire de fonctionnalités (pour le stockage des méthodes de publication et de recherche de services). Nous avons choisi d'utiliser le système de représentation de connaissances par objets AROM pour implémenter ce modèle. L'utilisation d'AROM afin d'opérationnaliser WSR-Model est directe étant donné qu'ils reposent tous les deux sur un formalisme orienté objet. Ainsi peu de modifications sont nécessaires (seuls les rôles sont ajoutés).

ProbCWS est un modèle de composition de service qui est utilisé comme base des applications de compositions de services Web. Ces dernières doivent reposer sur une architecture à trois composants (base de connaissances des problèmes, moteur d'inférence, et base de connaissances du domaine) étant donné que ProbCWS repose sur les méthodes de résolutions de problèmes. Nous avons choisi d'implémenter la base de connaissances des problèmes et le moteur d'inférence via le langage de résolution de problèmes associé au système AROM : AROMTasks. AROMTasks et ProbCWS se basant tous les deux sur les principes des méthodes de résolutions de problèmes à base de tâches, les mises en correspondance des concepts de base sont effectuées naturellement. Cependant, ProbCWS intègre le concept et la résolution de l'adaptation au contexte d'utilisation. Ceci nous a conduit à proposer une extension d'AROMTasks (AROMTasks-CWS) qui prend en compte :

- l'enrichissement de la définition des problèmes et des méthodes ;
- les instructions spécifiques d'appel (service concret, catégorie de services, et catégorie de compositions de services) dans le corps des méthodes.

Les deux chapitres suivant décrivent les implémentations de chacun de ces deux modèles. Le chapitre 10 décrit le registre de services Web WSR qui implémente le modèle WSR-Model. Le chapitre 11 définit la plate-forme de génération d'applications adaptées, GenAWS, qui implémente le modèle ProbCWS.

10 WSR : PROPOSITION D'UN REGISTRE DE SERVICES WEB

En vue de concevoir une application à base de services réutilisables, les clients n'ont pas, à ce jour, à leur disposition un registre de services efficace. De notre point de vue, un registre efficace est un registre qui propose des services au concepteur en adéquation avec le domaine de l'application à développer et les fonctionnalités à intégrer dans cette application. De plus, l'implémentation de l'application finale peut impliquer la prise en compte de l'adaptation au contexte d'utilisation. Par exemple, lors de l'implémentation de certaines applications mobiles, il est préférable que cette dernière s'adapte au dispositif utilisé et à la localisation de l'utilisateur. Nous avons, dans un premier temps, proposé un modèle de représentation de services (*cf.* chapitre 7) qui fournit quatre entités à prendre en compte lors de la description d'un service réutilisable (le service, le fournisseur, le domaine d'application, et le contexte d'utilisation). Cette représentation enrichie est la base du registre de services Web, WSR – *Web Services Registry* [Lopez-Velasco *et al.,* 2007a] [Lopez-Velasco *et al.,* 2007b], qui met en œuvre les fonctionnalités de publication (pour les fournisseurs de services), de recherche et de sélection (pour les clients de services).

Ce chapitre décrit le registre de services que nous proposons. Ce dernier implémente le modèle de représentation de services Web. Dans un premier temps, nous détaillons les différentes étapes de modélisation préalables à l'implémentation du registre de services, décrite dans la seconde section. Ensuite, nous illustrons l'utilisation du registre à travers une application Web qui fournit une interface (à base de formulaires) entre les utilisateurs du registre et le noyau fonctionnel. Un second accès à WSR est fourni à travers un ensemble de services Web qui permet aux acteurs logiciels d'interagir avec WSR. Enfin, nous donnons une évaluation de WSR par rapport aux solutions existantes en termes de registres de services Web.

10.1 Modélisation du registre

Cette première section décrit tout d'abord les objectifs du registre ainsi que les principaux cas d'utilisation et utilisateurs visés. Ensuite, les fonctionnalités offertes par le registre de services sont exposées.

10.1.1 Objectifs

Le cadre général des objectifs d'un registre a déjà été décrit dans le chapitre précédent (*cf.* section 9.1.1). Ici, nous nous préoccupons des objectifs spécifiques à notre cadre de travail : faciliter la tâche des utilisateurs lors de la génération d'applications adaptées au contexte d'utilisation à base de services Web.

La fonctionnalité de WSR est la même que celle des autres registres (stocker la représentation des services et fournir des moyens d'interroger les bases) et concerne deux acteurs du domaine des services Web : le fournisseur et le client.

Pour les fournisseurs. Les registres proposent aux fournisseurs de publier leurs services. La publication des services dans WSR repose sur le modèle de représentation de services Web. Cette publication permet de rendre visibles les services Web conçus par les fournisseurs. Ainsi, les services publiés peuvent être (ré)utilisés par des clients.

Pour les clients. Les registres de services Web fournissent aux clients un moyen de rechercher et de sélectionner des services Web préalablement publiés. La particularité de WSR est de permettre la recherche des services selon les besoins des clients en termes d'**implémentation d'applications adaptées au contexte d'utilisation**. Le concepteur établit sa demande de recherche de services selon le domaine d'application, les fonctionnalités désirées et le contexte d'utilisation auquel il souhaite que le système soit adapté. Le résultat de cette recherche est un ensemble de services répondant à chacune des fonctionnalités prédéterminées.

Selon l'utilisation d'un registre, ce dernier peut être de deux types : public ou privé.

Un **registre public** est visible et accessible par tous.

Un **registre privé** est seulement utilisé au sein d'une organisation ou plusieurs organisations. L'accès à ce type de registre est limité.

Nous avons choisi d'implémenter WSR comme un registre de type public. Son objectif est de rendre accessible au plus grand nombre les services disponibles afin d'augmenter leur taux de réutilisation.

10.1.2 Cas d'utilisation du registre de services

WSR propose à ses utilisateurs quatre types d'utilisation (*cf.* Figure 10.1) : la publication de services existants, la recherche de services élémentaires, la sélection de services élémentaires, et la recherche de services en vue de la conception d'un système particulier.

La publication de services. Pour le fournisseur de services Web, la publication réside dans le fait de renseigner des informations à propos du service selon le modèle de représentation de services Web supporté par le registre. Ces informations sont nécessaires lors de la recherche de services. Le fournisseur de services Web peut être un acteur humain, mais aussi un acteur logiciel, tel qu'un service Web de publication développé par des organisations de conception de services Web.

La recherche de services élémentaires. Les acteurs de la recherche sont les clients. Ils peuvent être de deux types : des acteurs humains ou des acteurs logiciels (tels que des services Web). Les critères de recherche de services élémentaires repose sur les informations composant le modèle de représentation de services Web.

La sélection de services élémentaires. À la suite d'une recherche de services, le client sélectionne, parmi l'ensemble des services renvoyés, celui qui lui convient le mieux.

La génération d'applications. L'acteur client peut aussi utiliser WSR afin de faire une recherche axée sur la génération d'applications particulières. Dans notre travail, cette recherche repose sur les critères du domaine d'application, des fonctionnalités à effectuer, et, si nécessaire, du contexte d'utilisation auquel doivent répondre les services. Le résultat de ce type de recherche est composé d'un ensemble de services Web.

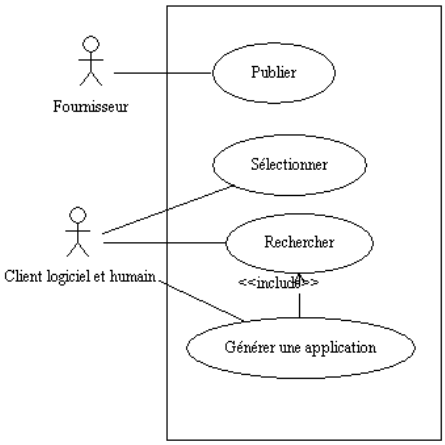


Figure 10.1 Cas d'utilisation du registre de services.

10.1.3 Fonctionnalités du registre

Nous mettons en évidence, dans cette section, les différentes fonctionnalités permises par WSR : la publication de services Web et la recherche.

10.1.3.1 Publication

La publication repose sur une séquence d'actions représentée par le diagramme de séquences UML (*cf.* Figure 10.2). Nous distinguons deux types d'action selon l'information qu'elles véhiculent : les actions à propos du choix de la catégorie du service et de la fonctionnalité du service, et les actions à propos de l'association du profil de déploiement et du contexte d'utilisation.

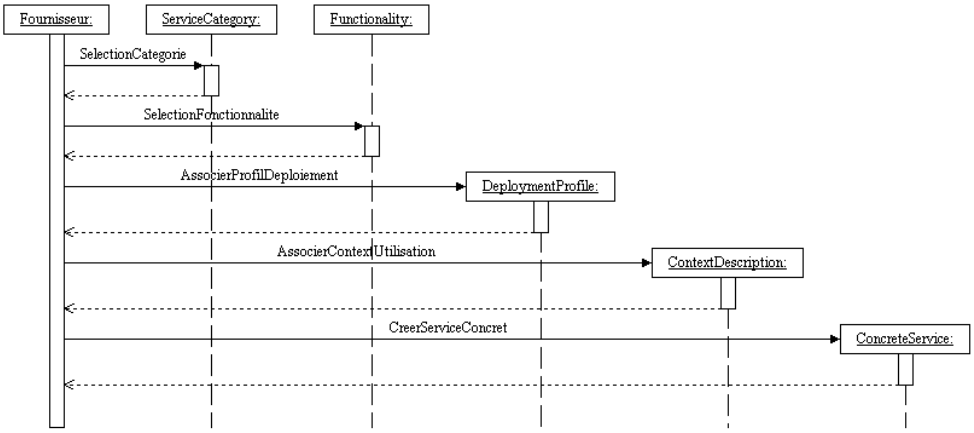


Figure 10.2 Diagramme de séquences UML représentant la publication d'un service à l'aide de WSR.

Le choix de la catégorie et la fonctionnalité du service à publier. La première étape de la publication est la sélection, par le fournisseur, de la catégorie à laquelle le service publié appartient et la fonctionnalité qu'il réalise. Le modèle de représentation de services Web intègre

des classes comprenant une description du domaine d'application (composée, entre autres, de la fonctionnalité du service) et une description du service (composée, entre autres, de la catégorie de services). Ces classes sont intentionnellement d'un haut niveau d'abstraction. Lors de la publication, le fournisseur peut :

- **sélectionner des instances existantes** (cas représenté par la Figure 10.2). Lors de l'implémentation du registre, nous avons instancié le modèle de représentation avec un ensemble de catégories de service et de fonctionnalités prédéterminées (telles que l'instance *geomatique* de la classe *Domain* ou la catégorie de services *Localization* de la classe *ServiceCategory*). Lors de la publication, le fournisseur peut alors sélectionner, parmi les instances existantes de catégorie et de fonctionnalité, celles lui convenant.
- **créer des instances** de catégorie de services et de fonctionnalités. Si aucune instance de catégorie et/ou de fonctionnalité ne convient aux besoins du fournisseur, nous lui laissons le soin de créer sa propre catégorie et/ou fonctionnalité. Ainsi, ceci contribue à l'évolutivité de l'ontologie du domaine.

L'association du profil de déploiement et du contexte d'utilisation au service Web publié. La particularité de la fonctionnalité de publication dans WSR est de proposer au fournisseur, si besoin est, d'associer au service publié un profil de déploiement et un contexte d'utilisation. Ces deux catégories d'information sont composées d'une description du contexte d'utilisation associée au package *UseContext* du modèle de représentation de services Web (*cf.* section 7.2.6).

10.1.3.2 Recherche

WSR propose deux types recherche : une recherche de services Web élémentaires et une recherche sous-jacente à la conception d'applications adaptées au contexte d'utilisation.

Recherche d'un service Web élémentaire. La recherche d'un service Web élémentaire dans WSR peut se faire selon le nom du service (comme le font les registres classiques de services Web). La particularité de WSR est qu'il propose aussi d'établir la recherche d'un service selon sa catégorie. Ainsi, la recherche est plutôt axée sur l'objectif du service et le besoin du client.

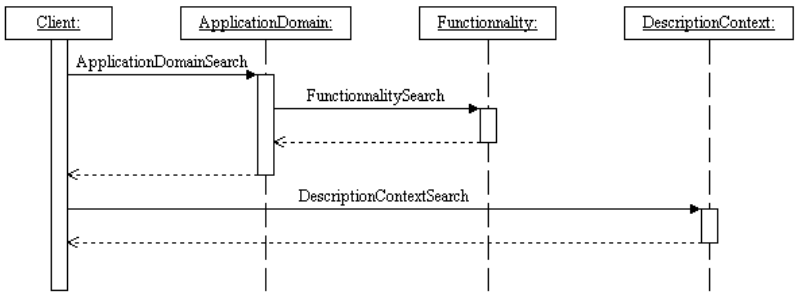


Figure 10.3 Diagramme de séquences UML représentant la recherche de services à l'aide de WSR.

Recherche sous-jacente à la conception d'une application adaptée au contexte d'utilisation. Le diagramme de séquences UML (*cf.* Figure 10.3) illustre la recherche proposée par WSR en vue de concevoir une application à base de services Web. Cette forme de recherche est basée sur trois critères : le domaine d'application auquel doit appartenir l'application finale, les fonctionnalités qui doivent être exécutées dans l'application, et le contexte d'utilisation auquel l'application doit être adaptée. Par l'intermédiaire du modèle de représentation de services Web, les services publiés dans WSR possèdent ce type d'information. Ainsi, à la suite de ce processus de recherche dans WSR, les services Web retournés correspondent aux critères demandés par le client (appartiennent à un domaine d'application spécifique, proposent une fonctionnalité

spécifique, et sont adaptés à un contexte d'utilisation). La composition des services renvoyés est réalisée dans la plate-forme de génération d'applications (*cf.* chapitre 11) selon le modèle de composition de services Web ProbCWS (*cf.* chapitre 8).

Le résultat d'une recherche quelle qu'elle soit est une liste de services Web. À ce niveau de notre travail de thèse, la sélection du service Web adéquat est alors à la charge du client et repose sur les critères composant la description des services Web.

10.2 Implémentation du registre

Nous présentons dans un premier temps l'architecture du registre de services Web WSR et argumentons par la suite les choix technologiques. Le fait que notre travail soit axé plus spécifiquement sur les services Web engendre, de notre point de vue, une mise en œuvre spécifique de la publication. Nous décrivons dans la dernière partie de cette section cette spécificité.

10.2.1 Architecture

L'objectif de WSR est de proposer, d'un côté, aux fournisseurs de publier leurs services, et de l'autre, aux clients d'en rechercher. Nous considérons les fournisseurs et les clients comme appartenant à la catégorie des utilisateurs de WSR. Un utilisateur peut être, soit humain, soit logiciel.

L'architecture de WSR est composée de deux parties : le noyau fonctionnel et l'interface (*cf.* Figure 10.4).

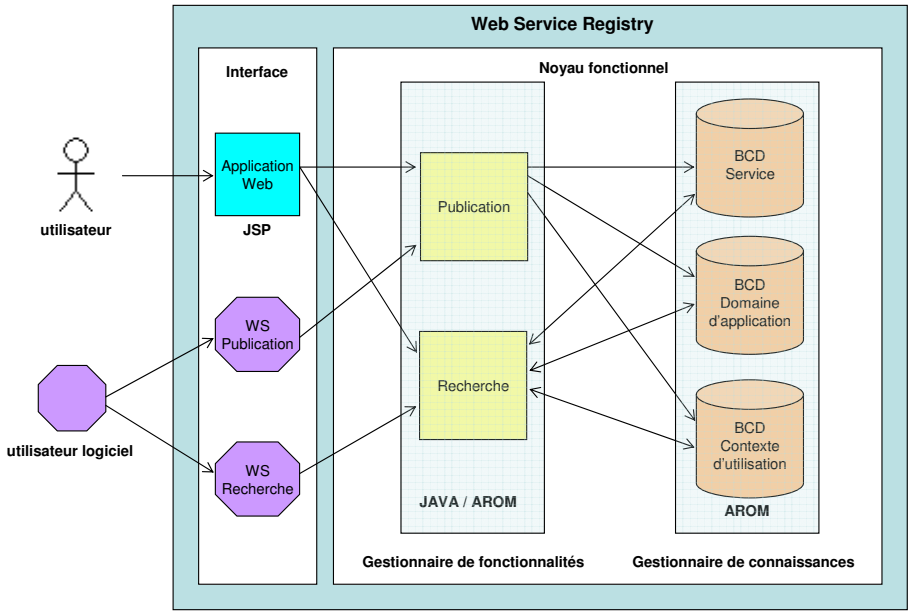


Figure 10.4 Architecture du registre de services WSR.

Le noyau fonctionnel. Le noyau fonctionnel est décomposé en deux parties : les gestionnaires de fonctionnalités et de connaissances. Le gestionnaire de fonctionnalités stocke les méthodes de publication et de recherche tandis que le gestionnaire de connaissances stocke les connaissances liées aux services Web (*cf.* section 9.2.1).

L'interface. Le module d'interface permet la communication entre les utilisateurs du registre et le noyau fonctionnel. Selon le type d'utilisateur (humain ou logiciel), nous avons mis en œuvre deux moyens adéquats de communiquer. Tout d'abord, nous avons implémenté une application Web qui est un moyen classique d'interaction entre un serveur et un utilisateur humain. Ensuite,

deux services Web (l'un pour la publication l'autre pour la recherche) réalisent l'interaction entre un utilisateur logiciel et le noyau fonctionnel de WSR.

Nous avons ici décrit l'architecture de WSR. L'étude de nos choix technologiques est décrit dans ce qui suit.

10.2.2 Choix technologiques

Afin d'implémenter le noyau fonctionnel et l'interface de l'architecture de WSR nous avons fait des choix technologiques que nous exposons ici.

10.2.2.1 Noyau fonctionnel

Le noyau fonctionnel permet de stocker les connaissances du domaine et les fonctionnalités fournies aux utilisateurs qui manipulent ces connaissances. De ce fait, ce composant de l'architecture comporte deux parties : le gestionnaire de connaissances et le gestionnaire de fonctionnalités. L'implémentation du noyau fonctionnel repose sur le système de représentation de connaissances par objets AROM. L'argumentation de ce choix est décrite dans le chapitre précédent (*cf.* section 9.1.2).

10.2.2.2 Interface

Le composant d'interface de l'architecture de WSR permet la communication entre les utilisateurs et le noyau fonctionnel du registre. Étant donné que le noyau fonctionnel de WSR est hébergé sur un serveur Web, l'interface de communication doit par conséquent permettre un accès à des fonctionnalités du Web. Nous avons choisi deux moyens d'accès au registre selon le type de l'utilisateur : une application Web ou des services Web.

Une application Web. Une application Web (formulaires HTML et pages *Java Server Pages* – JSP) permet la communication entre les utilisateurs humains et le noyau d'exécution.

Des services Web. Nous proposons deux services Web (un pour l'interface de chacune des fonctionnalités – publication et recherche) qui permettent la communication entre les utilisateurs logiciels (tels que d'autres services Web) et le noyau fonctionnel de WSR.

L'application Web et les services Web d'interface sont décrits en détail dans la section portant sur l'utilisation de WSR (*cf.* section 10.3.1).

10.2.3 Méthodes spécifiques à la publication de services Web : l'analyse d'une description WSDL

La technologie des services Web dépend du langage standard de description WSDL. Par conséquent, tout service Web doit posséder au préalable une description WSDL. Afin de faciliter la tâche du fournisseur lors de l'étape de publication, nous proposons dans WSR une analyse de la description WSDL qui évite au fournisseur de renseigner le profil fonctionnel du service publié, correspondant au package *Functional Profile* du modèle de représentation de services Web – *cf.* section 7.2.4.

Le processus d'analyse de la description WSDL des services publiés dans WSR est illustré par la Figure 10.5. Ce processus repose sur deux étapes : l'extraction et l'analyse des éléments de la description WSDL.

L'extraction. L'extraction des éléments du document WSDL a été implémentée par le biais de l'API wsdl4j[77] proposée par le *Java Community Process*[78]. Cette extraction permet d'obtenir le contenu de quatre éléments ou groupe d'éléments (`type`, `portType` et `operation`, `binding`, et `service`) que nous devons analyser lors de la publication.

L'analyse. Une fois que les éléments nécessaires à l'instanciation du modèle de représentation de services Web sont extraits, le module d'analyse crée les instances de la Base de Connaissances du Domaine Service AROM correspondantes :

- le contenu de l'élément `type` permet la création des instances `InParameter` et `OutParameter`.
- le contenu de l'élément `portType` et `operation` permet la création des instances `Method`.
- le contenu de l'élément `binding` permet la création des instances `Binding`.
- le contenu de l'élément `service` permet la création des instances `WebService`.

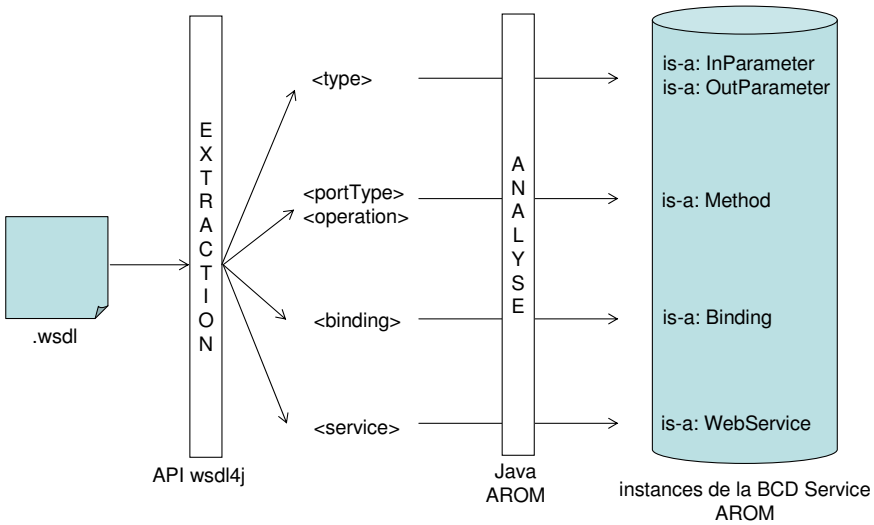


Figure 10.5 Processus d'extraction d'une description WSDL vers les instances d'une base de connaissances AROM selon le modèle de représentation de services Web.

Prenons l'exemple de la publication du service Web *GlobalWeather* dans WSR. La Figure 10.6 montre un extrait de la description WSDL de *GlobalWeather* et l'instance AROM issue du module d'analyse.

[77] L'API wsdl4j permet de faire l'extraction des éléments des fichiers WSDL issus de la version 1.1.

[78] http://www.jcp.org/

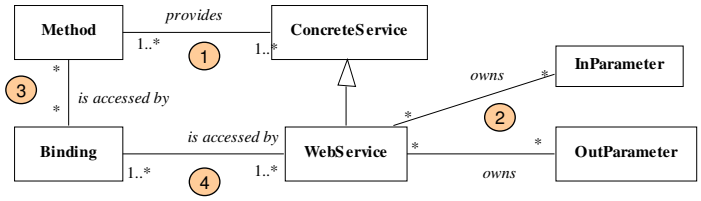
Figure 10.6 Analyse de la description WSDL du service Web *GlobalWeather* lors de sa publication dans WSR.

Le processus d'analyse permet non seulement de créer des instances mais aussi les associations existantes dans le modèle de représentation de services Web. La Figure 10.7 rappelle tout d'abord un extrait du modèle de représentation de services Web mettant en avant les éléments du profil fonctionnel (les classes *Method*, *InParameter*, *OutParameter*, et *Binding*) ainsi que leurs associations. Lors de la publication d'un service, le service concret (classe *Concrete Service*) et le service Web (classe *Web Service*) sont instanciés ainsi que les associations qu'elles possèdent avec les classes du package *Functional Profile*.

Selon l'exemple de l'analyse de la description WSDL réalisée lors de la publication du service *GlobalWeather* dans WSR, quatre associations (*tuple*) sont créées dans la base de connaissances du domaine Service (*cf.* Figure 10.7) :

- une association de type `Method2ConcreteService` qui permet de connaître pour le service publié les méthodes qu'ils proposent ;
- une association de type `InParameter2WebService` qui permet de connaître les types de paramètres nécessaires lors de l'invocation du service Web.
- une association de type `Binding2Method` qui permet de connaître le protocole d'accès de chaque méthode.
- une association de type `Binding2WebService` qui permet de connaître le protocole d'accès du service Web publié.

Extrait du modèle de représentation de services Web



Tuples correspondants à l'analyse de la description WSDL du service Web *GlobalWeather*

1
```
tuple:
is-a: Method2ConcreteService
 webService = _http___www_webserviceX_NET_GlobalWeather
 method = _http___www_webserviceX_NET_GlobalWeather_GlobalWeatherSoap_GetWeather
```

2
```
tuple:
is-a: InParameter2WebService
 inParameter = _http___www_webserviceX_NET_GlobalWeather_GlobalWeatherSoap_GetWeather_in_parameters
 webService = _http___www_webserviceX_NET_GlobalWeather
```

3
```
tuple:
is-a: Binding2Method
 binding = _http___www_webserviceX_NET_GlobalWeather_GlobalWeatherSoap
 method = _http___www_webserviceX_NET_GlobalWeather_GlobalWeatherSoap_GetWeather
```

4
```
tuple:
is-a: Binding2WebService
 webService = _http___www_webserviceX_NET_GlobalWeather
 binding = _http___www_webserviceX_NET_GlobalWeather_GlobalWeatherSoap
```

Figure 10.7 Associations déduites de l'analyse de la description WSDL du service Web *GlobalWeather* lors de sa publication dans WSR.

10.3 Utilisation de WSR

Selon le type d'utilisateur, il existe deux moyens d'utiliser WSR. Si l'utilisateur est un humain, nous mettons à sa disposition une application Web. Si l'utilisateur est de type logiciel (un service Web par exemple), nous proposons un ensemble de services Web qui permettent l'interface entre le noyau fonctionnel de WSR et ce type d'utilisateur.

10.3.1 Application Web

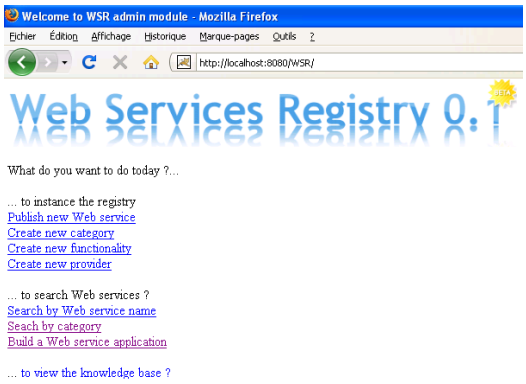


Figure 10.8 Page d'accueil de l'application Web de WSR.

Cette sous-section présente l'application Web implémentée pour l'utilisation du registre par un utilisateur humain. Lorsque cet utilisateur accède à l'application Web, il est accueilli par une première

page (*cf.* Figure 10.8) par l'intermédiaire de laquelle il peut accéder aux deux principales fonctionnalités de WSR : la publication ou la recherche de services.

10.3.1.1 Publication

La publication d'un service Web se fait par l'intermédiaire d'une page de l'application Web comportant un formulaire de publication. Nous distinguons deux types d'information que l'utilisateur doit renseigner lors de la publication : les aspects fonctionnels et le contexte d'utilisation.

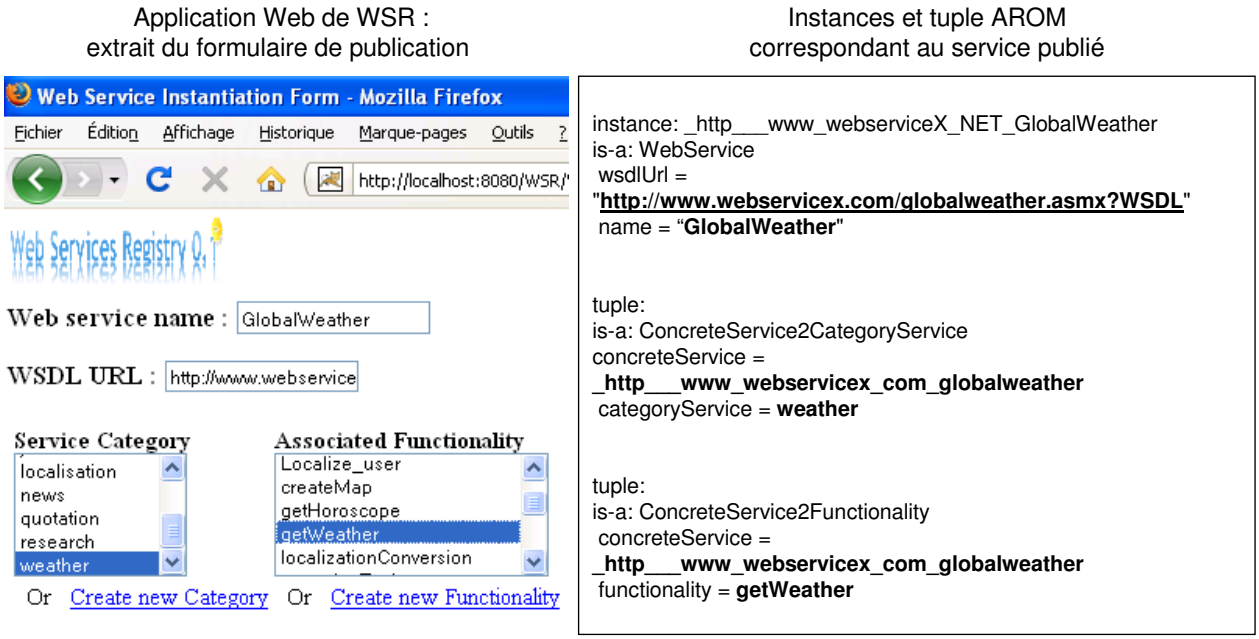


Figure 10.9 Illustration de l'utilisation de l'application Web concernant la publication d'un service Web dans WSR.

Les aspects fonctionnels. Les informations que le fournisseur doit renseigner à propos des aspects fonctionnels du service publié sont : le **nom du service**, **la localisation de la description WSDL** dans le cas où il publie un service Web, la **catégorie du service**, et la **fonctionnalité** qu'il réalise.

- **Le nom du service**. Le fournisseur renseigne le nom du service Web qu'il publie, par l'intermédiaire duquel le service sera identifié dans WSR.
- **La localisation de la description WSDL**. La description WSDL récupérée est ensuite analysée par la méthode décrite dans la section 10.2.3.
- **La catégorie et la fonctionnalité du service**. Le fournisseur doit sélectionner parmi les catégories et les fonctionnalités pré-existantes dans WSR celles qui conviennent à leur service. Si aucune instance existante ne convient, le fournisseur a à sa disposition la possibilité de créer une nouvelle catégorie et une nouvelle fonctionnalité.

La Figure 10.9 illustre la publication via l'application Web des aspects fonctionnels du service Web *GlobalWeather*. À gauche de la figure, nous pouvons voir un extrait de l'application Web impliquant la partie du formulaire de publication concernant les aspects fonctionnels (nom du service, localisation de la description WSDL, catégorie et fonctionnalité associées du service).

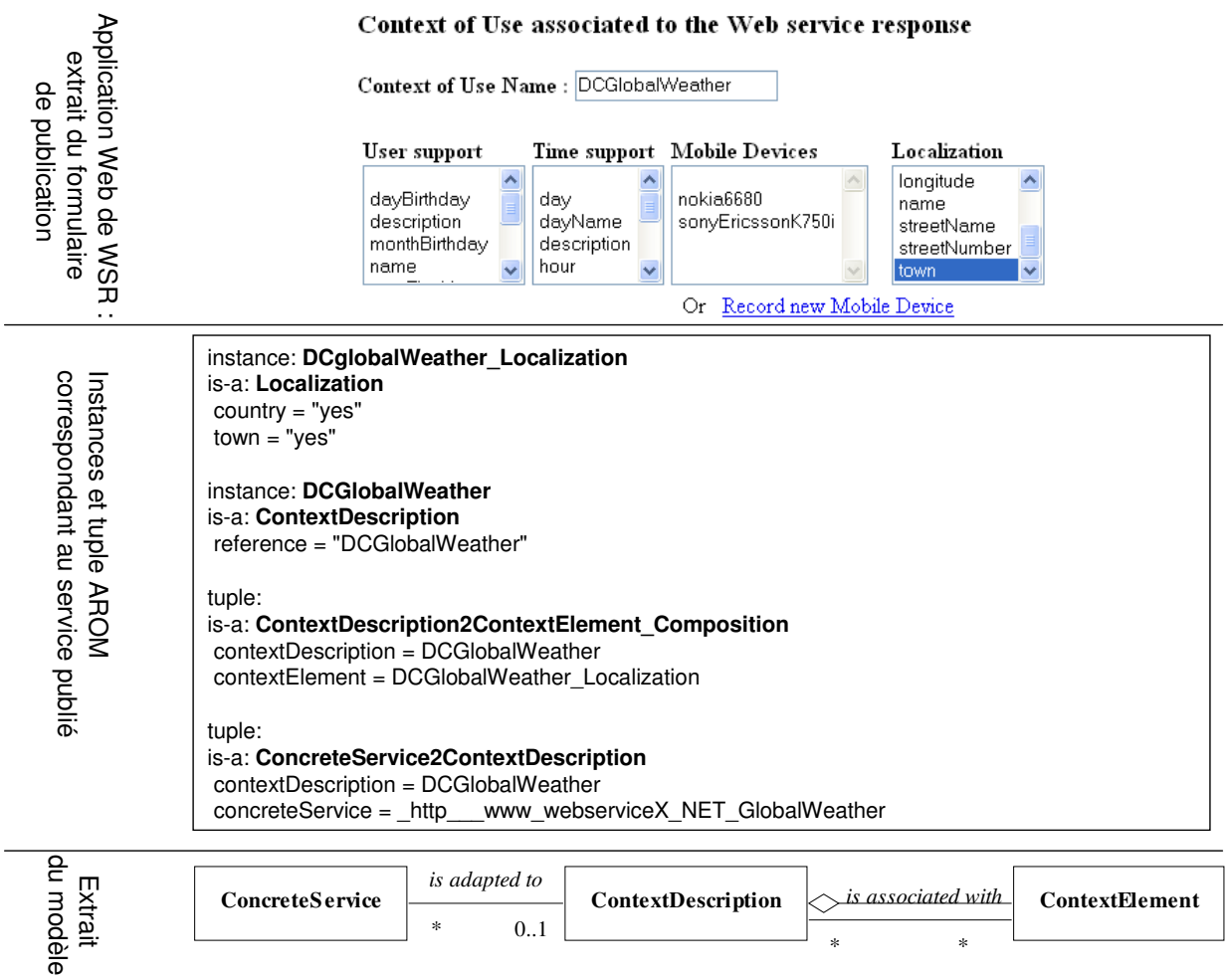


Figure 10.10 Illustration de l'utilisation de l'application Web concernant l'association du contexte d'utilisation au service à publier dans WSR.

Le contexte d'utilisation. Par l'intermédiaire du formulaire de publication de l'application Web de WSR, le fournisseur a à sa disposition un moyen d'associer un contexte d'utilisation au service qu'il va publier. L'originalité de WSR réside en l'association du contexte d'utilisation au service publié. Cette association peut être de deux sortes et permet de :

- **décrire les contraintes d'exécution** du service et,
- de **représenter le contexte d'utilisation** auquel le service peut s'adapter.

Le fournisseur a à sa disposition, par l'intermédiaire du formulaire de publication, l'ensemble des attributs de la représentation du contexte d'utilisation (*cf.* Figure 10.10). Le fournisseur sélectionne alors les attributs du contexte qui lui conviennent. La Figure 10.10 illustre l'association du contexte *DCGlobalWeather* à la publication du service Web *GlobalWeather*. Ce contexte d'utilisation est composé de deux éléments (*country* et *town*) du contexte issus de la classe *Localization*.

10.3.1.2 Recherche

La recherche de services dans WSR repose sur trois formulaires (*i.e.* trois types de recherche) : une recherche classique par nom du service, une recherche par catégorie de services, et une recherche en vue de concevoir une application adaptée au contexte d'utilisation.

La recherche par nom du service. WSR propose classiquement la recherche par nom de service (*cf.* Figure 10.11 à gauche). La méthode sous-jacente à la recherche par nom permet aussi une recherche par mots-clés, c'est-à-dire que l'item recherché appartient au nom du service. Dans l'exemple représenté dans la Figure 10.11, le client cherche un service nommé *weather* ou dont le nom comporte *weather*.

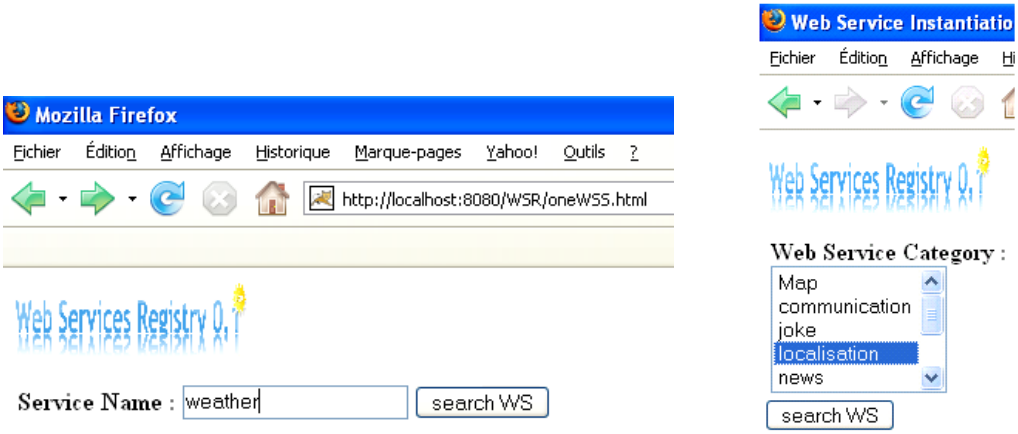


Figure 10.11 Recherche d'un service Web élémentaire : à gauche par nom de service, à droite par catégorie de services.

La recherche par catégorie de services. Le client peut faire une recherche selon la catégorie de services. Nous proposons à l'utilisateur une page Web où apparaissent l'ensemble des catégories représentées dans la base de connaissances. Le client n'a plus qu'à sélectionner la catégorie désirée et les services associés à cette dernière sont affichés. Dans l'exemple illustré par la Figure 10.11 (à droite), le client recherche un service appartenant à la catégorie *localisation*.

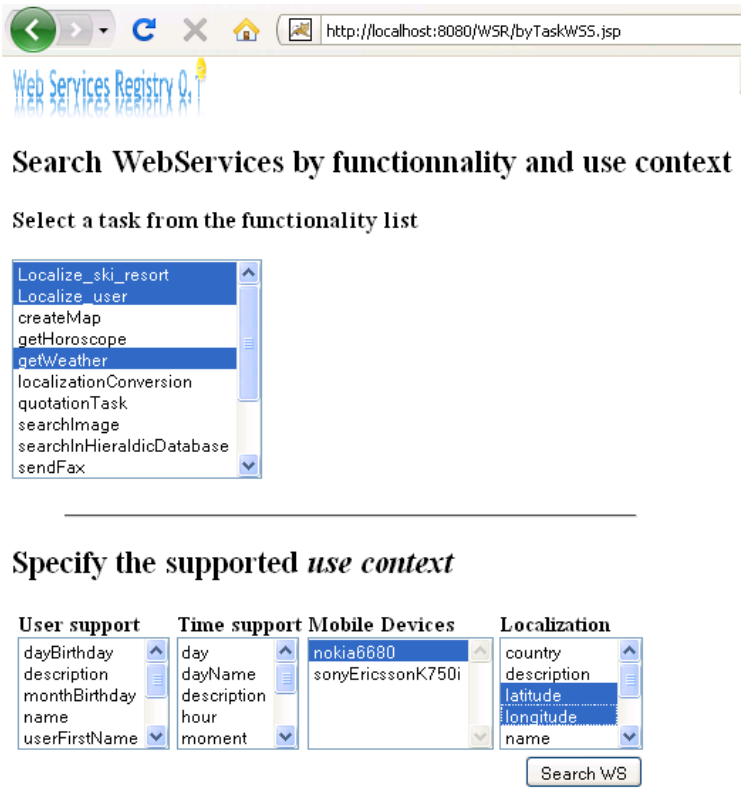


Figure 10.12 Recherche en vue de concevoir une application adaptée au contexte d'utilisation.

La recherche pour la conception d'applications adaptées au contexte d'utilisation. La dernière forme de recherche proposée par WSR constitue les prémisses à la conception d'applications adaptées au contexte d'utilisation à base de services Web. Cette recherche inclut les fonctionnalités que l'application doit réaliser et le contexte d'utilisation pour lequel le client souhaite que l'application soit adaptée. Par l'intermédiaire d'une page Web, le client choisit, parmi l'ensemble des fonctionnalités présentes dans la base de connaissances, la fonctionnalité recherchée, puis choisit les attributs du contexte d'utilisation auquel il souhaite une adaptation (*cf.* Figure 10.12). Dans l'exemple illustré par la Figure 10.12, le client recherche des services en vue d'implémenter une application qui réalisera des fonctionnalités de localisation de station de ski (*Localize_ski_resort*), de l'utilisateur (*Localize-user*) et d'obtention des conditions météorologiques (*getWeather*). De plus, le client souhaite que l'application puisse s'adapter au dispositif de l'utilisateur de type *nokia6680*, et à sa localisation (la *latitude* et la *longitude*).

À la suite d'une recherche dans WSR, le résultat est une liste de services Web comprenant dans leur description détaillée :

- leur **aspect fonctionnel** (nom, méthodes proposées, paramètres d'entrée et de sortie),
- les **aspects concernant leur fonctionnalité** (catégorie de services, fonctionnalité réalisée) et,
- si c'est le cas, la description des **éléments du contexte d'utilisation** auxquels ils sont adaptés.

Cette description sous-jacente au modèle de représentation de services Web permet aux clients de sélectionner le ou les services qu'ils préfèrent.

Lorsque l'utilisateur est un client humain, nous avons choisi intentionnellement de laisser le processus de sélection à ce dernier. Ainsi ce processus peut reposer sur des critères qui lui sont propres (tels que le fournisseur, le contexte d'utilisation, les types de paramètres).

Lorsque l'utilisateur est un agent logiciel, la liste de résultat est une liste ordonnée de services Web selon leur pertinence. Les services sont tout d'abord classés selon la (ou les) fonctionnalité(s) qu'ils proposent. Ensuite, dans les sous-ensembles (représentant les services Web d'une fonctionnalité en particulier) les services Web sont classés selon leur pertinence en termes de contexte d'utilisation. Nous utilisons un système de pondération afin de déterminer cette pertinence. Ce système de pondération repose sur les **éléments du contexte d'utilisation** pour lequel le service est adapté. La pertinence des services Web retrouvés à l'issue de la recherche est la somme des poids *Pcu* calculés pour chaque élément du contexte d'utilisation pour lequel le service est adapté. Les éléments du contexte d'utilisation sont composés d'un ensemble de critères. Par exemple, l'élément dispositif est composé du critère taille de l'écran et mémoire vive. Le poids *Pcu* de ces éléments est calculé de la manière suivante :

$$Pcu = \frac{cr}{cd}, \text{ où } cr = \{critères\ retrouvés\} \text{ et } cd = \{critères\ désirés\}.$$

Pcu appartient à l'ensemble [0;1] où 1 représente le poids le plus important. La liste des services est ainsi hiérarchisée selon la somme des poids obtenus.

10.3.2 Services Web d'interface

Nous décrivons ici l'ensemble des services Web d'interface proposés afin de permettre aux utilisateurs de type logiciel de WSR d'utiliser ses fonctionnalités. Nous comptons deux services Web principaux : celui de publication et celui de recherche. Un autre service Web, celui d'interrogation de

la base de connaissances, est nécessaire pour la réalisation des deux services Web d'interface. L'ensemble des signatures des méthodes proposées par les services sont disponibles dans l'annexe 2.

10.3.2.1 Service Web d'interrogation de la base de connaissances

Le service Web d'interrogation permet de renvoyer à l'utilisateur l'ensemble des catégories et des fonctionnalités présentes dans la base de connaissances du domaine de WSR. Ces connaissances sont utiles aux utilisateurs de WSR dans deux cas :

- lors de la **publication de services Web**. Les fournisseurs de services Web doivent prendre connaissance des instances de catégories et de fonctionnalités présentes dans la base de WSR afin d'enrichir la représentation des services à publier.
- lors de la **recherche de services Web**. Les clients prennent connaissance des catégories et des fonctionnalités des services enregistrés dans WSR afin d'appeler l'une ou l'autre des méthodes de recherche.

La description WSDL de ce service, nommé *WSRDomainKnowledge*, possède trois méthodes : une méthode d'interrogation des **catégories de service**, une méthode d'interrogation **des fonctionnalités des services**, d'une méthode d'interrogation des **éléments du contexte d'utilisation**.

L'interrogation des catégories. La méthode *CategoryRequest* permet de connaître l'ensemble des instances de catégories de services existantes dans la base de connaissances du Service de WSR.

L'interrogation des fonctionnalités. Deux méthodes permettent de connaître l'ensemble des instances des fonctionnalités de services présentes dans la base de connaissances du Service de WSR.

- La première méthode, nommée *FunctionalityRequest*, renvoie toutes les fonctionnalités de la base de connaissances.
- La seconde méthode, nommée *FunctionalityRequestByDomain*, renvoie les fonctionnalités qui sont associées à un domaine d'application spécifique.

L'interrogation du contexte d'utilisation. Le service Web *WSRDomainKnowledge* permet aussi de connaître les éléments du contexte d'utilisation prédéterminés dans la base de connaissances du contexte d'utilisation de WSR. Une première méthode, nommée *UseContextRequest*, permet de connaître l'ensemble des éléments du contexte d'utilisation prédéterminés dans la base de connaissances. Une seconde méthode, nommée *UseContextElementRequest*, permet de connaître les valeurs constituant un élément du contexte d'utilisation en particulier.

En retour aux trois méthodes d'interrogation, le service Web renvoie un fichier de type RDF représentant l'ontologie constituée par l'ensemble des instances demandées de la base de connaissances du domaine. Ainsi, les utilisateurs de WSR peuvent prendre connaissance des catégories et des fonctionnalités des services et des contextes d'utilisation présents dans le registre. L'utilisation d'ontologies en tant que résultat de l'interrogation du registre, permet de mettre en œuvre la fonctionnalité de recherche via les alignements d'ontologies.

10.3.2.2 Service Web de publication

Le service Web de publication, nommé *WSPublish*, est directement accessible par les utilisateurs afin qu'ils publient dans WSR leurs services Web.

La publication d'un service Web repose sur quatre méthodes : la publication des aspects **fonctionnels**, la publication des **objectifs du service**, la publication des **contraintes d'exécution**, et la publication **du contexte d'utilisation**.

La publication des aspects fonctionnels. Cette méthode est nommée *AddFunctionalDescription*. La publication des aspects fonctionnels du service est constituée de la publication de quatre éléments de la représentation du service : son nom, la localisation de la description WSDL, le prix du service, et des commentaires du fournisseur. À la suite de la publication des aspects fonctionnels d'un service, le service Web de publication renvoie à l'utilisateur une clé avec laquelle il peut continuer la publication d'autres aspects du service (la fonctionnalité, les contraintes d'exécution, et le contexte d'utilisation).

La publication des objectifs du service. Cette méthode est nommée *AddNonFunctionalDescription*. La publication des objectifs du service consiste à renseigner la (ou les) catégorie(s) auxquelles le service appartient et la (ou les) fonctionnalité(s) qu'il réalise.

La publication des contraintes d'exécution. Cette méthode est nommée *AddExecutionConstraintDescription*. Les contraintes d'exécution sont représentées par des éléments du contexte. La méthode de publication des contraintes d'exécution consiste à renseigner les attributs des éléments du contexte pour lequel le service a des contraintes d'exécution.

La publication du contexte d'utilisation. Cette méthode est nommée *AddUseContextDescription*. La publication du contexte d'utilisation permet d'associer à un service Web le contexte d'utilisation auquel le service est adapté. Lors de l'appel de la méthode de publication du contexte d'utilisation, l'utilisateur renseigne les éléments du contexte d'utilisation et les attributs pour lequel le service est adapté.

Si l'utilisateur ne connaît pas les instances de catégorie, de fonctionnalité, des éléments du contexte d'utilisation et leurs attributs existants dans la base de connaissances, il doit, au préalable, faire appel au service Web d'interrogation (*cf.* paragraphe précédent).

Nous ajoutons à ce service Web deux méthodes de **création d'instances** :

- La première, nommée *categoryRecord*, est nécessaire si l'utilisateur souhaite ajouter de nouvelles catégories dans la base de connaissances de WSR.
- La seconde, nommée *functionalityRecord*, est nécessaire si l'utilisateur souhaite ajouter de nouvelles fonctionnalités dans la base de connaissances de WSR.

10.3.2.3 Service Web de recherche

Le service Web de recherche permet d'implémenter la fonctionnalité de recherche de WSR pour les utilisateurs logiciels.

Le service Web de recherche, nommé *WSRSearch*, est composé de trois méthodes, proposant chacune un type de recherche : La recherche par **mots-clés**, la recherche par **catégorie de services**, et la recherche pour **concevoir une application adaptée au contexte d'utilisation** selon les fonctionnalités à accomplir et le contexte d'utilisation à prendre en compte.

Les résultats des appels des méthodes de recherche diffèrent selon le type de recherche :

Recherche par mots-clés. Cette méthode est nommée *byKeywordSearch*. Le résultat de l'appel à la méthode de recherche par mots-clés est l'ensemble des services Web dont le nom contient le(s) mot(s)-clé(s) demandé(s).

Recherche par catégorie. Cette méthode est nommée *byCategorySearch*. Le résultat de l'appel à la méthode de recherche par catégorie est l'ensemble des services Web de la catégorie demandée.

Recherche pour la conception d'applications. Cette méthode est nommée *conceptionSearch*. Le résultat de l'appel à la méthode de conception d'applications est une liste ordonnée de services Web selon leur pertinence (*cf.* section 10.3.1.2).

10.4 Évaluation de WSR

Nous pouvons comparer et évaluer WSR par rapport aux autres registres de services Web selon quatre axes : la description enrichie des services, la recherche de services Web pour la conception d'applications, l'utilisation du registre, et la mise en œuvre de l'adaptation.

La description enrichie des services Web. Lors de la **publication** de services Web, les fournisseurs complètent la description fonctionnelle de leurs services (permise par WSDL) avec de nouvelles catégories d'information (telles que le domaine d'application ou le contexte d'utilisation) issues du modèle de représentation de services Web. Cet enrichissement de la représentation de services Web permet de faciliter l'étape de **sélection** étant donné que les clients ont à leur disposition plus d'informations pour établir leur choix.

La recherche de services Web pour la conception. L'originalité de WSR par rapport aux autres registres de services Web est qu'il propose une recherche particulière de services Web. Cette recherche permet d'aider les clients à rechercher des services en vue de la **conception d'applications adaptées au contexte d'utilisation orientées services Web**. Cette recherche repose sur les critères suivants : les **fonctionnalités** que l'application doit réaliser et le **contexte d'utilisation** auquel l'application doit être adaptée.

L'utilisation de WSR. L'utilisation des registres existants repose, soit sur une application Web, pour les registres publics, soit sur des spécifications en vue d'implémenter les registres privés. WSR étant un registre public, nous avons choisi d'implémenter une application Web. Afin que des utilisateurs de type logiciel accèdent à WSR nous mettons aussi à disposition des **services Web d'interface** permettant d'utiliser les fonctionnalités de WSR (publication et recherche) par d'autres services Web.

La mise en œuvre de l'adaptation. Dans le chapitre de l'état de l'art portant sur la mise en œuvre de l'adaptation et les services Web (chapitre 5), et le chapitre de synthèse de l'état de l'art (chapitre 6), nous avons mis en évidence un ensemble de types et techniques d'adaptation. WSR met en œuvre **l'adaptation de la demande service** par **la technique de sélection de services Web**. Ainsi WSR respecte le critère d'évaluation **d'adaptation minimale** puisqu'il propose la résolution de l'adaptation lors de la phase de modélisation d'une application à base de services Web.

10.5 Conclusion

Le registre que nous proposons offre des fonctionnalités à la fois aux fournisseurs et aux clients de services Web.

WSR pour les fournisseurs de services Web. WSR propose aux fournisseurs de **publier** leurs services, lors de la phase préliminaire du cycle de vie d'une application orientée services. La publication dans WSR rassemble l'ensemble des informations du modèle de représentation de services Web. Ainsi, les services Web stockés dans WSR possèdent une description détaillée qui

couvre les aspects fonctionnels et non fonctionnels, et qui caractérise le contexte d'utilisation auquel le service est adapté.

WSR pour les clients de services Web. WSR propose aux clients de services Web, lors de la phase de modélisation d'une application orientée services, de **rechercher** les services convenant à leurs besoins. WSR fournit trois types de recherche :

- une **recherche par mots-clés.**
- une **recherche par catégorie de services.**
- une recherche selon les **fonctionnalités** à réaliser et le **contexte d'utilisation** pour lequel le client souhaite une forme d'adaptation. Ce type de recherche permet de mettre en œuvre l'étape de recherche lors de la **conception d'applications orientées services Web adaptées au contexte d'utilisation**.

À la suite de la recherche, le client sélectionne le (ou les) service(s) Web qui lui convien(en)t le mieux parmi la liste des services Web retrouvés. Cette sélection repose sur la description des services Web, basée sur le modèle de représentation de services Web.

Afin d'établir la communication entre les utilisateurs et le noyau fonctionnel de WSR nous proposons deux moyens d'accès à WSR selon le type d'utilisateur : une application Web et un ensemble de services Web.

L'application Web. L'application Web propose aux utilisateurs humains d'utiliser les fonctionnalités de publication et de recherche de WSR.

Les services Web d'interface. WSR met à la disposition des utilisateurs logiciels (tels que de services Web) deux services Web d'interface (**service Web de publication** et **service Web de recherche**) et un service Web d'interrogation des catégories, des fonctionnalités, et des éléments du contexte d'utilisation renseignés dans WSR.

WSR met en œuvre une partie des étapes de la phase de modélisation du cycle de vie des applications à base de services. Le résultat des recherches faites par l'intermédiaire de WSR est une liste de services Web. Afin d'utiliser les services Web retrouvés, nous proposons une plate-forme de génération d'applications adaptées au contexte d'utilisation orientée services Web. Le chapitre suivant décrit cette plate-forme qui valide, entre autres, le modèle de composition de services Web ProbCWS.

11 GENAWS : UNE PLATE-FORME DE GÉNÉRATION D'APPLICATIONS ADAPTÉES AU CONTEXTE D'UTILISATION À BASE DE SERVICES WEB

Nous avons proposé un registre de services Web, WSR, qui met en œuvre trois étapes du cycle de vie d'une application à base de services : la **publication** par les fournisseurs de services, et les étapes de **recherche** et de **sélection** de services par les clients. Notre registre s'appuie sur le modèle de représentation de services Web, WSR-Model.

L'étape qu'il reste à mettre en œuvre dans notre proposition est la **composition de services Web**. Avec l'augmentation du nombre de services Web disponibles sur le Web, il est difficile d'analyser et de générer manuellement une composition de services Web. Ceci a pour conséquence une activité accrûe en matière de recherche et développement de la composition automatique de services Web [Sheth *et al.*, 2003]. Cependant, aucune solution existante ne prend en compte l'ensemble des problématiques inhérentes à la composition de services Web : la définition de la composition, la recherche et la sélection des services Web composants, le contrôle de l'exécution de la composition et son évolutivité.

La définition de la composition. Afin de définir de manière plus aisée la composition de services Web, nous pensons, à l'instar de [Wagner *et al.*, 2002], qu'il est préférable de décomposer la requête de l'utilisateur en sous-objectifs élémentaires. Prenons l'exemple d'un utilisateur nomade qui souhaite voir afficher une carte sur son dispositif mobile avec les restaurants les plus proches de l'endroit où il se situe (la requête de l'utilisateur). Cette tâche complexe, incluant de l'adaptation au contexte d'utilisation (au dispositif mobile, au profil de l'utilisateur et à sa localisation), est constituée d'appels à plusieurs services Web spécialisés. Nous considérons alors la composition comme étant une tâche complexe. Dans notre travail, le résultat de cette dernière résulte d'une décomposition en un ensemble de tâches simples pouvant être résolues par un service Web élémentaire.

La recherche et la sélection de services Web composants. Une fois la composition définie, il faut rechercher et sélectionner les services Web permettant de mettre en œuvre l'ensemble des tâches simples.

Le contrôle de l'exécution de la composition de services Web. Le contrôle de l'exécution met en œuvre l'étape de surveillance de la composition de services Web. Lors de cette étape, il faut, à chaque invocation de services Web composants, prévoir de faire face à certains problèmes qui pourraient survenir tels que l'indisponibilité d'un service ou l'échec de l'appel d'un service.

L'évolutivité de la composition. Lorsque le contrôle de l'exécution de la composition de services Web révèle une anomalie (telle que l'indisponibilité du service Web) il faut alors faire évoluer la composition. Cette évolutivité consiste à trouver des solutions telles que l'invocation de nouveaux services, ou l'invocation d'une nouvelle définition de composition. Cette activité requiert d'intégrer à la composition de services Web des techniques d'adaptation au contexte d'utilisation (telles que la compensation) et se traduit par la nécessité de mettre en œuvre une composition dynamique de services Web.

Afin de prendre en compte l'ensemble des étapes décrites ci-dessus lors de la conception d'applications à base de services Web, nous proposons la plate-forme GenAWS (*Generating Adapted applications based on Web Services*). Cette plate-forme rassemble l'ensemble des processus nécessaires afin d'utiliser au mieux des services Web existants dans le cadre du développement d'applications à Architecture Orientée Services adaptées au contexte d'utilisation. La réalisation de cette plate-forme repose sur les modèles et l'application préalablement définis (le modèle de représentation de services, WSR-Model – chapitre 7 ; le modèle de composition de services Web, ProbCWS – chapitre 8 ; et le registre de services Web, WSR – chapitre 10).

Ce chapitre s'articule autour de trois axes permettant de décrire GenAWS : tout d'abord la modélisation de la plate-forme, ensuite l'étude des choix d'implémentation, et enfin un aperçu de l'utilisation de GenAWS. Nous déterminons ce chapitre par l'évaluation de GenAWS par rapport aux applications de composition de services Web existantes.

11.1 Modélisation de la plate-forme GenAWS

Cette première section concerne l'étude des objectifs de cette plate-forme, la mise en évidence des principaux cas d'utilisation et la description des fonctionnalités fournies par GenAWS.

11.1.1 Objectifs

Le rôle de la plate-forme GenAWS est d'accompagner les acteurs intervenant dans le cycle de vie des services Web (fournisseur et client) afin de faciliter l'utilisation de cette technologie dans la mise en œuvre d'une application à base de services.

Cette plate-forme prend en charge les activités liées aux services Web, de leur publication à leur exécution, dans le cadre d'une composition de services Web. GenAWS prend en compte l'ensemble des activités suivantes : la publication, la recherche et la sélection de services Web, la description de la composition, ainsi que le contrôle et l'évolutivité de l'exécution de la composition. L'ensemble de ces activités permet la **génération d'applications à base de services Web adaptées au contexte d'utilisation**. L'adaptation au contexte d'utilisation dans GenAWS est intégrée à ces quatre premières activités.

11.1.2 Cas d'utilisation de la plate-forme GenAWS

Cette sous-section met en relief les différentes utilisations possibles de GenAWS et les acteurs qui peuvent interagir avec cette plate-forme. La Figure 11.1 illustre l'ensemble des cas d'utilisation selon les différents acteurs : le fournisseur, le client, le concepteur, et le moteur d'inférence.

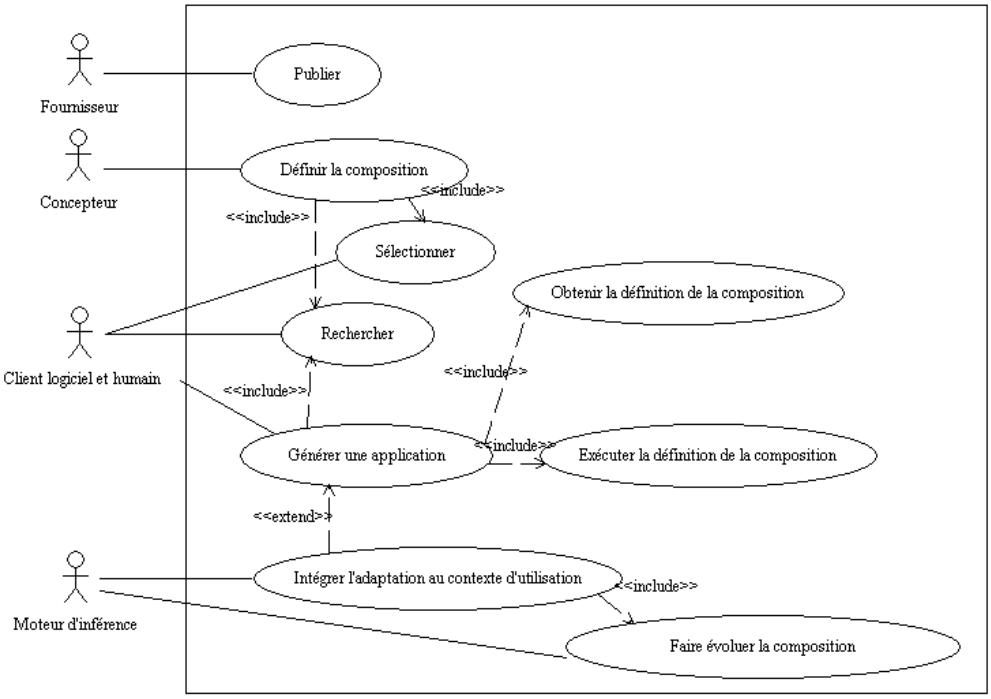


Figure 11.1 Cas d'utilisation de la plate-forme GenAWS.

Utilisation de GenAWS par les fournisseurs. La première activité du cycle de vie d'une application à base de services Web est la publication des services Web par le fournisseur. Dans GenAWS, cette activité est sous-jacente à un appel à WSR (*cf.* chapitre 910).

Utilisation de GenAWS par les concepteurs. Nous appelons *concepteur* un type particulier de client. Lors de la définition d'une composition dans GenAWS, le concepteur instancie un nouveau problème dans la base de connaissances des problèmes de la plate-forme. La définition de la composition repose sur le modèle ProbCWS (*cf.* chapitre 8). L'activité *Définir la composition* inclut les activités de *recherche* et de *sélection* de services Web de base.

Utilisation de GenAWS par les clients. Un client peut être soit humain, soit logiciel. Il utilise GenAWS dans trois cas : pour *rechercher* et *sélectionner* des services Web et pour *générer une application*. Ces activités sont inhérentes à un appel à WSR (*cf.* chapitre 10). Dans GenAWS, l'activité de *génération d'applications* contient des spécificités telles qu'*obtenir la définition de la composition* de services Web composant l'application ou exécuter directement l'application à générer (N.B. ces spécificités sont décrites de manière plus détaillée plus bas – *cf.* section 11.1.3).

Rôle du moteur d'inférence dans GenAWS. La particularité de GenAWS repose sur deux points. Tout d'abord, notre proposition permet de générer des applications adaptées au contexte d'utilisation, ensuite GenAWS propose de faire évoluer une composition de services Web. Ces particularités sont rendues possibles par l'intermédiaire du modèle de représentation de services Web (*cf.* chapitre 7) et du modèle de composition de services Web ProbCWS (*cf.* chapitre 8). Afin de réaliser ces aspects, GenAWS intègre un moteur d'inférence qui permet :

- de sélectionner les résolutions de problèmes répondant à la requête du client,
- de choisir les services Web à invoquer pour exécuter au mieux la résolution.

11.1.3 Fonctionnalités de la plate-forme

Seule la fonctionnalité de **génération d'applications adaptées** est décrite ici. Les autres fonctionnalités sont, soit celles du registre WSR (telles que la publication, la recherche et la sélection de services Web), soit décrite dans le chapitre concernant ProbCWS – *cf.* chapitre 8 (en ce qui concerne la définition de la composition).

La génération d'applications est composée de deux sous-fonctionnalités : **obtenir la composition** et **exécuter la composition**. Le client peut choisir l'une ou l'autre de ces sous-fonctionnalités. Dans le cas de l'exécution de la composition, la définition de la composition est transparente au client.

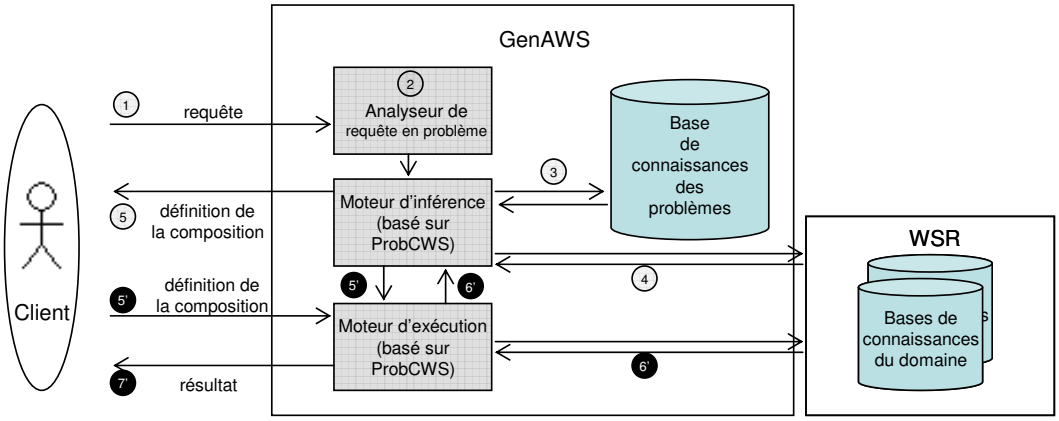


Figure 11.2 Illustration des transferts de données dans le cadre de la génération d'applications avec GenAWS.

L'obtention de la **définition de la composition** est réalisée en cinq étapes (représentées par des disques gris dans la Figure 11.2) :

Étape 1. La première étape consiste en l'**envoi de la requête** par le client à la plate-forme. Cette requête est composée d'un ensemble de paramètres permettant à GenAWS de mettre en œuvre la génération d'applications. Ces paramètres sont les suivants :

- **La (ou les) catégorie(s) de services Web** auxquels le client souhaite que les services Web composant l'application appartiennent.
- **La (ou les) fonctionnalité(s)** que les services Web doivent exécuter.
- **Le contexte d'utilisation** auquel l'application générée doit être adaptée.
- **Les types d'entrée et de sortie** de l'application. Si le client souhaite exécuter l'application, il renseigne en plus la valeur des paramètres d'entrée.

L'ensemble des catégories de services Web, de fonctionnalités, et les éléments du contexte d'utilisation disponibles est stocké dans les bases de connaissances du domaine de WSR.

Étape 2. La seconde étape consiste à **traduire la requête du client en problème**. GenAWS se base pour cela sur le modèle de composition de services Web ProbCWS. Une fois traduite en problème selon ce modèle, la requête pourra être traitée par des mécanismes de résolution.

Étape 3. Le problème correspondant à la requête est transmis par l'analyseur de requête au moteur d'inférence. Ce dernier **recherche un problème équivalent dans la base de connaissances des problèmes**. Ainsi, si un problème équivalent à la requête du client existe dans la base, GenAWS suit la résolution du problème stockée dans la base de connaissances des problèmes. Si aucun problème équivalent n'est trouvé dans la base de connaissances des problèmes, la plate-forme propose au client de définir sa propre composition.

Étape 4. La quatrième étape consiste à **rechercher les services Web ou les catégories de services Web** dans les bases de connaissances du domaine afin d'implémenter chaque étape de la résolution du problème (*i.e.* chaque étape de la composition de services Web).

Étape 5. La cinquième et dernière étape consiste à **renvoyer le résultat** au client. Le client peut obtenir en retour de la requête différentes formes de définition de la composition :

- Le client peut obtenir une définition de l'enchaînement de la composition n'incluant que des **appels de services Web de base**.
- Le client peut obtenir une définition de l'enchaînement de la composition incluant des **appels de services Web et de catégories de services Web**.
- Le client peut obtenir une définition de l'enchaînement de la composition incluant des **appels de services Web, de catégories de services Web, et de compositions de services Web**.

Le résultat de la définition de la composition est donné sous la forme d'une résolution de problèmes ProbCWS.

La seconde sous-fonctionnalité de la génération d'applications est l'**exécution de la composition.** La demande de l'exécution peut être de deux formes (*cf.* étape 5', Figure 11.2[79]) :

Le client ne connaît pas a priori la définition de la composition. Par conséquent les quatre premières étapes de l'obtention de la définition décrite plus haut doivent être réalisées et le moteur d'inférence renvoie ensuite le résultat de la définition de la composition au moteur d'exécution. Dans ce cas, le moteur d'inférence transmet au moteur d'exécution une définition de la composition constituée de services Web de base.

Le client connaît au préalable la définition de la composition. Cette définition peut être transmise par le client lui-même au moteur d'exécution afin d'être exécutée.

L'exécution de la composition consiste ensuite à réaliser chaque étape de la résolution du problème donnée par la définition de la composition ProbCWS. Il existe trois types d'étapes de résolution (étape 6' – *cf.* Figure 11.2) :

L'étape de la composition est représentée par un service Web. Dans ce cas, le moteur d'exécution invoque le service Web connu par l'intermédiaire de WSR. Si le service est indisponible, il met en œuvre une adaptation par compensation.

L'étape de la composition est représentée par une catégorie de services Web. Dans ce cas le moteur d'exécution sélectionne un service Web issu de la catégorie de services Web d'après le mécanisme décrit dans le chapitre précédent (*cf.* section 10.3.2.3). Cette sélection est réalisée via le service Web d'interface de recherche de WSR (*cf.* section 10.3.2.3).

L'étape de la composition est représentée par une composition de services Web. Dans ce cas, le moteur d'exécution renvoie la demande de résolution de la composition au moteur d'inférence. La définition de cette composition est réalisée classiquement (*cf.* étapes 1 à 4) et est ensuite transmise au moteur d'exécution.

À la suite de l'exécution, la plate-forme renvoie le résultat au client (étape 7' – *cf.* Figure 11.2).

[79] Les étapes spécifiques à l'exécution de la composition sont illustrées par des disques noirs dans la Figure 11.2.

11.2 Implémentation de la plate-forme

Cette section présente l'implémentation de la plate-forme GenAWS. Nous décrivons dans un premier temps l'architecture de la plate-forme, puis mettons en évidence nos choix technologiques. Enfin, les méthodes spécifiques inhérentes à nos choix technologiques sont décrites.

11.2.1 Architecture

L'objectif de GenAWS est de proposer aux clients (humain ou logiciel) un moyen de générer des applications adaptées au contexte d'utilisation à base de services Web.

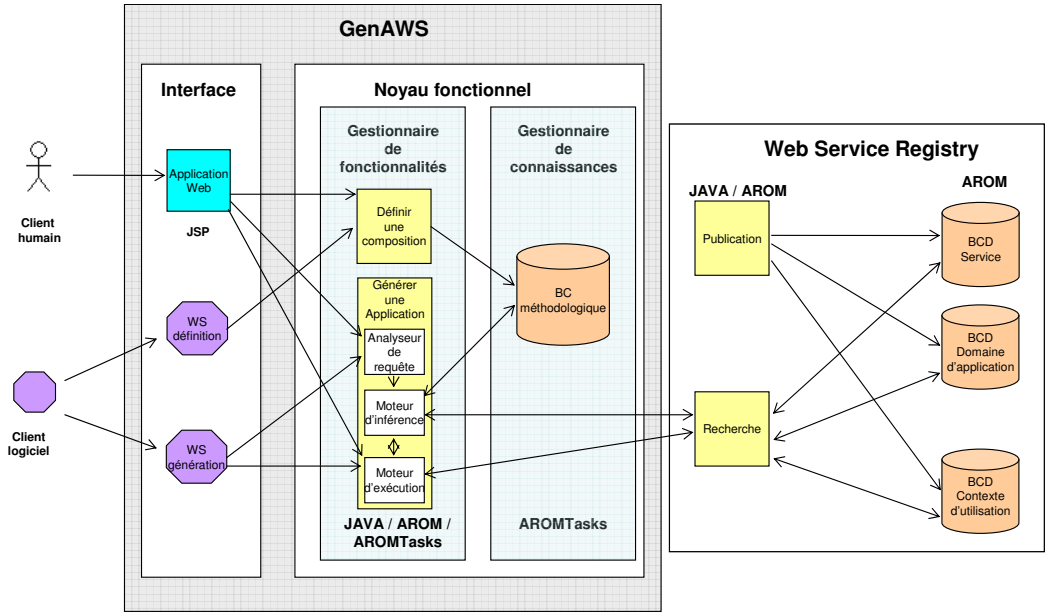


Figure 11.3 Architecture de la plate-forme GenAWS.

L'architecture de cette plate-forme (*cf.* Figure 11.3) est composée de deux parties : le noyau fonctionnel et l'interface.

Le noyau fonctionnel. Le noyau fonctionnel de GenAWS est hébergé sur un serveur et repose sur deux gestionnaires : le gestionnaire de fonctionnalités et le gestionnaires de connaissances.

- **Le gestionnaire de fonctionnalités** est composé de trois modules. Le module de *définition de la composition* permet de mettre en œuvre la fonctionnalité d'instanciation de la base de connaissances des problèmes (appelée *base de connaissances méthodologique*). Le *moteur d'inférence* et le *moteur d'exécution* permettent de mettre en œuvre la génération d'applications.
- **Le gestionnaire de connaissances** permet de stocker la base de connaissances méthodologique. Cette base gère les connaissances correspondant aux problèmes et à leur résolution sous la forme de ProbCWS.

L'interface. Cette partie de l'architecture permet de faire l'interface entre les clients de la plate-forme (qu'ils soient acteurs humain ou logiciel) et les fonctionnalités fournies par GenAWS mises en œuvre par le noyau fonctionnel. Selon le type d'utilisateur, la plate-forme propose deux types d'interface : une application Web ou des services Web. L'application Web est le moyen le plus classique pour mettre en œuvre l'interaction entre des utilisateurs humains et un serveur. Afin d'établir la communication entre les acteurs logiciels et le noyau fonctionnel, GenAWS

possède deux services Web pour les deux fonctionnalités principales proposées (définition de compositions et génération d'applications).

11.2.2 Choix technologiques

L'implémentation des composants de l'architecture de GenAWS repose sur des technologies spécifiques à l'opérationnalisation des modèles (*cf.* chapitre 9). Nous argumentons tout d'abord nos choix à propos des technologies choisies pour mettre en œuvre le noyau fonctionnel, puis les choix sous-jacents à la réalisation de l'interface.

11.2.2.1 Noyau fonctionnel

Le noyau fonctionnel a pour objectif de stocker les connaissances des problèmes et de mettre en œuvre les fonctionnalités de GenAWS qui utilisent ces connaissances. Il est décomposé en deux parties distinctes : le gestionnaire de connaissances et le gestionnaire de fonctionnalités. Nous avons choisi d'héberger le noyau fonctionnel de GenAWS sur un serveur Web afin qu'il soit accessible par le plus grand nombre.

Étant donné que GenAWS utilise des fonctionnalités de WSR nous devons choisir des technologies compatibles. L'implémentation de WSR étant basé sur le système de représentation de connaissances par objets AROM, notre choix de technologie pour l'implémentation de GenAWS se tourne naturellement vers le langage AROMTasks [Genoud *et al.*, 2005], associé au système de représentation de connaissances par objets AROM (*cf.* section 9.2). AROMTasks étant un langage de résolution de problèmes à base de tâches, il peut ainsi implémenter la fonctionnalité de génération d'applications de GenAWS basé sur ProbCWS. AROMTasks rend possible la résolution de problèmes en exploitant les connaissances du domaine à partir des connaissances méthodologiques. AROMTasks a déjà fait ses preuves dans d'autres domaines, tels que la bio-informatique [Chabalier, 2004].

Afin de prendre en compte les spécificités de ProbCWS (telles que la prise en compte de l'adaptation au contexte d'utilisation), nous avons proposé une extension d'AROMTasks nommée AROMTasks-CWS (*cf.* section 9.2.4).

Le détail de l'implémentation de chacun des gestionnaires composant le noyau fonctionnel de GenAWS est décrit ci-dessous :

Le gestionnaire de connaissances. Le gestionnaire de connaissances est composé d'une base de connaissances des problèmes. Cette base stocke les problèmes et leur(s) résolution(s). Ce module est implémenté à l'aide du langage AROMTasks-CWS. La base de connaissances est alors appelée base de connaissances méthodologique.

Le gestionnaire de fonctionnalités. Le gestionnaire de fonctionnalités est composé de deux modules mettant en œuvre les fonctionnalités fournies par GenAWS.

- **La définition de la composition**. Ce module permet d'instancier la base de connaissances méthodologique lors de la définition de composition exprimée en AROMTasks-CWS par les concepteurs. L'implémentation de ce module est basée sur le langage Java et le système de représentation de connaissances par objets AROM.
- **La génération d'applications**. Ce module réalise la fonctionnalité de génération d'applications proposée par GenAWS. Différents composants implémentent ce module de génération d'applications : l'*analyseur de requête*, le moteur d'inférence, et le moteur d'exécution (cf. section 11.1.3 et Figure 11.2 pour une description détaillée de cette fonctionnalité). L'*analyseur de requête* traduit la requête de l'utilisateur en problème. Ce

composant est implémenté à l'aide du langage Java. Une fois la requête de l'utilisateur exprimée et les services Web décrits, idéalement, un moteur de planification doit trouver une collection de services Web qui répond à la requête. Ce rôle est tenu par le *moteur d'inférence* implémenté à l'aide d'AROMTasks-CWS. Une fois le problème résolu, le *moteur d'exécution* exécute la composition qui vient d'être définie. L'implémentation du moteur d'exécution est basée principalement basé sur le langage AROMTasks-CWS.

L'utilisation d'AROMTasks-CWS au sein du moteur d'exécution apporte un avantage indéniable, celui de la **surveillance de l'exécution**. L'exécution de l'application, suivant la définition de la composition préalablement déterminée par le moteur d'inférence, est réalisée par AROMTasks-CWS. AROMTasks-CWS a la particularité de vérifier la validité de l'exécution à chacune des étapes sous-jacentes à la résolution du problème. Si le résultat ne convient pas (ou si dans notre cas le service Web n'est pas disponible) AROMTasks-CWS effectue un retour en arrière dans l'exécution. Dans GenAWS, AROMTasks-CWS permet de réaliser **l'évolutivité de l'exécution de la composition des services Web**.

11.2.2.2 Interface

L'interface de GenAWS permet de mettre en œuvre la communication entre les utilisateurs de la plate-forme et son noyau fonctionnel. Le noyau fonctionnel de WSR étant hébergé sur un serveur Web nous avons choisi deux modes de communication Web qui diffèrent selon le type d'utilisateur de GenAWS : un application Web et des services Web.

Une application Web. L'implémentation de l'application Web repose sur un formulaire HTML et des pages JSP. Cette application permet la communication entre les utilisateurs humains et le noyau fonctionnel de GenAWS.

Des services Web. L'interface de GenAWS propose deux services Web permettant aux utilisateurs logiciels (tels que d'autres services Web) d'utiliser les fonctionnalités de GenAWS. Le module d'interface contient un service Web de définition de compositions et un service Web de génération d'applications.

Une description détaillée de ces deux types d'interface est présente dans la section 11.3.

11.3 Utilisation de la plate-forme GenAWS

Nous fournissons deux moyens d'utiliser la plate-forme GenAWS selon le type de l'utilisateur : une application Web pour les utilisateurs humains et un ensemble services Web d'interface pour les utilisateurs de type logiciel. Nous décrivons ces deux moyeux d'utilisation de la plate-forme dans ce qui suit.

11.3.1 Application Web

Par l'intermédiaire de la page d'accueil de l'application Web de GenAWS, le client peut utiliser les fonctionnalités, soit de WSR (*cf.* à gauche de la Figure 11.4), soit spécifiques à GenAWS (*cf.* à droite de la Figure 11.4). Nous nous intéressons ici plus particulièrement aux fonctionnalités spécifiques de GenAWS. À partir de cette page d'accueil, le client accède aux pages de définition de nouvelles compositions ou aux deux pages de génération d'applications (obtenir la définition de la composition ou exécuter l'application).

Figure 11.4 Page d'accueil de l'application Web de GenAWS.

11.3.1.1 Définition de la composition

L'instanciation de nouvelles définitions de composition se fait par l'intermédiaire d'une page spécifique de l'application Web (*cf.* Figure 11.5).

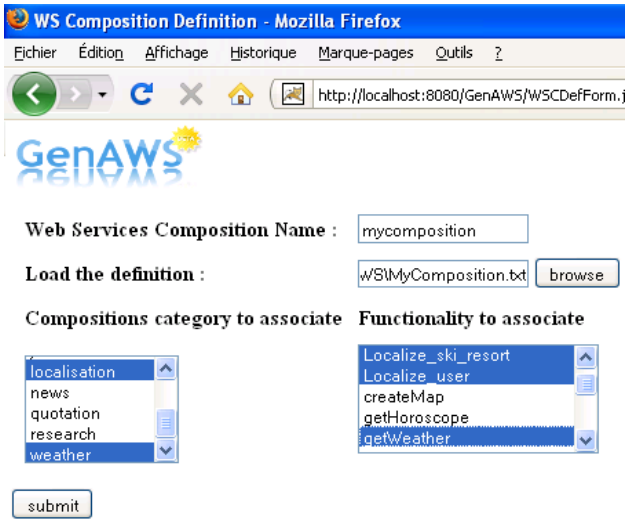


Figure 11.5 Formulaire de l'application Web de GenAWS pour l'enregistrement d'une définition de composition définie en ProbCWS.

La définition de la composition utilisée afin d'instancier la base de connaissances méthodologique est définie par le client en AROMTasks-CWS.

Par l'intermédiaire de la page Web, le client renseigne tout d'abord le nom de la composition, la localisation locale du fichier de description ainsi que la catégorie et les fonctionnalités associées à la composition. Les listes de catégories et de fonctionnalités sont dynamiquement liées à la base de connaissances du domaine de WSR. L'ajout de nouvelles catégories ou de nouvelles fonctionnalités se fait par l'intermédiaire de ce registre (*cf.* section 10.3.1.1).

Dans l'exemple d'instanciation de définition (*cf.* Figure 11.5), la composition est nommée *myComposition* et est associée à deux catégories de compositions (*localisation* et *weather*) et trois fonctionnalités (*localize_ski_resort*, *localize_user*, et *getWeather*). L'ordre d'invocation des services Web, appelés par les tâches, est décrit dans la définition de la composition ProbCWS renseignée par le client.

11.3.1.2 Génération d'applications

La plate-forme GenAWS fournit aux clients deux fonctionnalités de génération d'applications :

- la plate-forme renvoie, selon la requête du client, la **définition de la composition** exprimée en ProbCWS, décrivant la mise en œuvre de l'application demandée ;
- la plate-forme renvoie le résultat de l'**exécution de l'application** déterminée par le biais de la requête du client.

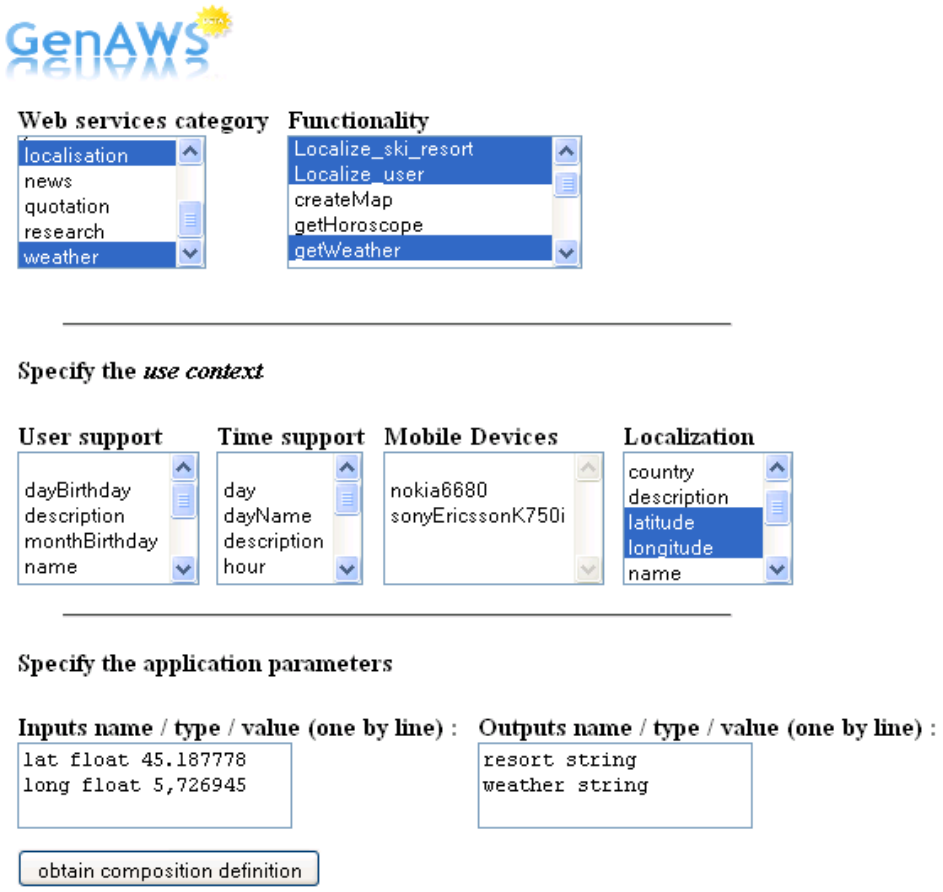


Figure 11.6 Illustration de la page Web de génération d'applications au sein de l'application Web de GenAWS.

Chacune de ces fonctionnalités est accessible via une page spécifique de l'application Web. Afin de spécifier la requête du client via l'application Web de GenAWS, les formulaires proposent aux clients de spécifier :

- les **catégories de compositions** et les **fonctionnalités** associées à l'application qu'ils désirent ;
- le **contexte d'utilisation** pour lequel les clients souhaitent que l'application soit adaptée ;

- les **noms et les types des paramètres** d'entrée et de sortie. Pour la fonctionnalité d'exécution de l'application, le client doit aussi spécifier la **valeur de chaque paramètre d'entrée**.

La Figure 11.6 illustre la demande d'un client de l'exécution d'une application via la page Web spécifique. Le client souhaite l'exécution d'une application associée à deux catégories de compositions (`localisation` et `weather`) et trois fonctionnalités (`localize_ski_resort`, `localize_user`, et `getWeather`). L'application doit être adaptée à la latitude et la longitude de l'utilisateur. L'application prend en paramètre d'entrée deux variables (`lat` et `long`) de type `float` et renvoie deux paramètres de sortie (`resort` et `weather`) de type `string`. Le résultat de cette fonctionnalité de génération d'application est par conséquent deux valeurs de type `string` représentant les stations de ski proches de l'utilisateur et la météo dans ces stations.

À la suite de la génération d'applications, s'il existe plusieurs compositions qui puissent répondre à la requête du client, la plate-forme établit une liste hiérarchisée de ces compositions selon leur pertinence. Nous utilisons un système de pondération afin de déterminer la pertinence des compositions retrouvées. Ce système de pondération s'appuie sur les éléments sur lesquels repose la recherche de composition (la catégorie de compositions, les fonctionnalités à exécuter, et les éléments du contexte d'utilisation pour lesquels le client souhaite de l'adaptation). La pertinence des compositions est la somme des poids *Pcomp* calculés pour chaque élément de recherche. Chaque élément peut être consitué de plusieurs éléments (par exemple, l'élément de recherche *catégorie* peut être composé de deux critères – *localisation* et *weather*). Le poids *Pcomp* de ces éléments est calculé de la manière suivante :

$$Pcomp = \frac{cr}{cd}$$, où $cr = \{critères\ retrouvés\}$ et $cd = \{critères\ désirés\}$.

Pcomp appartient à l'ensemble [0;1] où 1 représente le poids le plus important.

Lorsque le client souhaite la définition d'une composition, la plate-forme lui renvoie la liste des compositions répondant à sa requête hiérarchisée selon le poids obtenu. Les compositions sont représentées par leur nom et les éléments les représentant (catégorie, fonctionnalité, et contexte d'utilisation). Si le client souhaite par la suite exécuter une des compositons, il doit le faire via la plate-forme GenAWS afin de pouvoir mettre en œuvre les techniques d'adaptation.

Lorsque le client souhaite exécuter directement une application, la plate-forme fait appel à la composition qui a obtenu le poids le plus important.

11.3.2 Services Web d'interface

Nous proposons dans GenAWS deux types de services Web qui permettent l'utilisation de la plate-forme à des clients de type logiciel (tels que d'autres services Web). Le premier type de service Web permet l'instanciation de nouvelles définitions de composition dans la base de connaissances méthodologique. Le second type de service Web permet l'utilisation de la fonctionnalité de génération d'applications adaptées au contexte d'utilisation. L'ensemble des signatures des méthodes proposées par les services sont disponibles dans l'annexe 3.

11.3.2.1 Service Web de définition d'une composition

Le service Web de définition d'une composition permet à un client logiciel d'ajouter une nouvelle résolution de problèmes au sein de la base de connaissances méthodologique.

Le service Web `RecordWSComposition` possède une méthode unique du même nom. Cette méthode prend en paramètre le nom de la définition de la composition à enregistrer, la localisation du

fichier texte contenant la définition de la composition exprimée en ProbCWS ainsi que la catégorie de services Web (`WSCategory`) et les fonctionnalités associées à la définition de la composition. À la suite de l'appel, le service Web renvoie un booléen afin de rendre compte au client de la bonne exécution de la méthode d'enregistrement du problème dans la base de connaissances.

11.3.2.2 Service Web de génération d'applications

Le service Web de génération d'applications, nommé `ApplicationGenerator`, propose deux méthodes : la première permet d'obtenir la définition de la composition permettant d'exécuter l'application, la seconde permet d'exécuter directement l'application.

Obtenir la définition de la composition. La méthode `AppliGenDef` du service Web prend en paramètre la requête du client exprimée en ProbCWS. Dans l'expression de la requête, seule la tâche principale est définie. Le résultat de l'invocation de cette méthode est la définition de la composition répondant à la requête du client.

Exécuter l'application. La méthode `AppliGenExec` prend en paramètre la définition de la composition à exécuter, exprimée en ProbCWS, et les noms et les valeurs des paramètres d'entrée stockés dans une structure de type clé-valeur (*Map*). La définition de la composition peut être celle renvoyée en résultat de l'invocation de la première méthode (`AppliGenDef`). Le résultat de la méthode `AppliGenExec` est une structure de type clé-valeur (*Map*) contenant les noms et les valeurs des paramètres de sorties de l'application exécutée par la plate-forme.

11.4 Évaluation de GenAWS

GenAWS se distingue des autres plates-formes de composition de services Web selon quatre points : la transversalité sous-jacente à la plate-forme, les fonctionnalités proposées aux utilisateurs, la mise en œuvre de l'exécution, et l'intégration de l'adaptation au contexte d'utilisation.

La transversalité sous-jacente à GenAWS. La plate-forme GenAWS a la particularité de mettre en œuvre un ensemble d'étapes du cycle de vie d'une application orientée services. Les étapes de *publication*, *recherche* et *sélection* sont réalisées par l'intermédiaire d'appels à WSR. Les étapes de *définition de la composition* et *d'exécution* sont réalisées par le noyau fonctionnel propre à GenAWS.

Les fonctionnalités fournies par GenAWS. L'implémentation de la composition de services Web dans GenAWS est assimilée à la génération d'applications. GenAWS propose aux utilisateurs, soit d'obtenir la définition de la composition, soit de l'exécuter.

La mise en œuvre de l'exécution. La fonctionnalité de générations d'applications de GenAWS donne la possibilité aux utilisateurs d'obtenir le résultat de leurs exécutions. Cette exécution est permise via le langage AROMTasks. Ce langage comporte l'avantage de surveiller l'exécution de la résolution de problèmes. Lors de la mise en œuvre de la composition de services Web si l'exécution d'un service Web échoue, GenAWS invoque un service Web équivalent (c'est-à-dire appartenant à la même classe que le service qui a échoué).

L'intégration de l'adaptation au contexte d'utilisation. Étant donné que les fonctionnalités de la plate-forme GenAWS se basent sur le modèle de composition ProbCWS, la plate-forme hérite de la mise en œuvre de l'adaptation au contexte d'utilisation permise par ce modèle. Nous rappelons brièvement les formes et techniques d'adaptation permise par ProbCWS :

- L'**adaptation du résultat** par le biais des techniques par conversion et par déploiement de services Web d'adaptation ;

- l'**adaptation de la demande de services Web** par le biais des techniques par compensation et par sélection de services Web ;
- l'**adaptation de la composition** par le biais des techniques par compensation, par sélection de services Web, et par sélection de compositions de services Web.

11.5 Conclusion

La plate-forme GenAWS propose aux fournisseurs un moyen de publier leurs services et des définitions de composition, et aux clients de rechercher, de sélectionner et de composer les services Web répertoriés au sein de la plate-forme afin de générer une application adaptée au contexte d'utilisation.

La mise en œuvre de GenAWS repose sur un ensemble d'application, modèle, et langages :

L'intégration de l'application WSR. Afin d'intégrer les fonctionnalités de publication, de recherche et de sélection de services Web, GenAWS inclut dans son architecture le registre de services Web WSR préalablement réalisé (*cf.* chapitre 10).

La mise en œuvre du modèle ProbCWS. Nous avons réalisé un modèle de composition de services Web, ProbCWS, qui permet de définir une composition comme une résolution de problèmes (*cf.* chapitre 8). Ce modèle permet d'intégrer l'adaptation au contexte d'utilisation et de mettre en œuvre la compensation de services Web. Les deux fonctionnalités proposées par GenAWS (définition de compositions et génération d'applications) reposent sur le modèle ProbCWS.

L'utilisation du langage AROMTasks. Afin de mettre en œuvre le modèle de composition de services Web ProbCWS sur lequel se base GenAWS nous avons choisi un langage de résolution de problèmes par modèles de tâches. Le langage AROMTasks est un bon candidat puisqu'il repose sur le système de représentation de connaissances par objets AROM, système de base de WSR. En choisissant AROMTasks, nous facilitons la communication entre le noyau fonctionnel propre à GenAWS et celui de WSR.

CONCLUSION

12 BILAN ET PERSPECTIVES

12.1 Bilan du travail réalisé

À la lumière de l'étude des travaux portant sur les services Web (travaux sur la description, publication, recherche et sélection, composition, et adaptation), aucune solution ne propose aux acteurs interagissant avec les services Web (fournisseurs et clients) un cadre de travail incluant la plupart des processus nécessaires à l'utilisation des services Web (publication, recherche, sélection, et composition) tout en intégrant l'adaptation au contexte d'utilisation.

Afin d'atteindre ce but, nous avons proposé un modèle de représentation de services (WSR-Model) et un modèle de composition de services Web (ProbCWS) reposant respectivement sur une description à base d'ontologies et sur la résolution de problèmes. Ces modèles ont été validés par deux prototypes : le registre de services Web (WSR) valide le modèle de représentation, et la plate-forme de gestion de services Web (GenAWS) valide le modèle de composition de services Web.

La particularité de notre travail par rapport aux autres propositions du domaine des services Web repose en deux points. Tout d'abord notre proposition prend part à la plupart des étapes du cycle de vie d'une application à base de services. Ensuite, la mise en œuvre de l'adaptation au contexte d'utilisation est prise en compte de manière transversale tout au long du processus de conception d'applications à base de services. Ainsi nous contribuons à améliorer les étapes de description, de publication, de recherche, de sélection, de composition de services Web, et enfin de mise en œuvre de l'adaptation au contexte d'utilisation dans le domaine des services Web.

Description. Le modèle de représentation de services Web que nous proposons repose sur des formalismes orientés objet. Ceci permet au modèle d'être extensible et facilement intégrable au sein de projets externes à notre travail de thèse. Le modèle de représentation de services Web a pour objectif de faciliter des étapes au cycle de vie d'une application orientée services telles que la publication, recherche et sélection de services Web. La particularité de ce modèle est qu'il intègre en ensemble de catégories d'information indépendantes qui décrivent les services Web. Ces catégories sont les suivantes :

- la description des **aspects fonctionnels du service** (cette catégorie est incluse au sein du standard WSDL),
- la description du **domaine d'application** auquel s'applique le service,

- la description des **aspects non fonctionnels** du service (tels que le fournisseur, le profil de déploiement, et les contraintes d'exécution),
- la description du **contexte d'utilisation** auquel d'adapte le service.

Publication, recherche et sélection. Ces trois étapes du cycle de vie d'une application à base de services, bien qu'elles ne soient pas réalisées par les mêmes acteurs (le fournisseur publie, tandis que le client recherche et sélectionne), s'appuient sur la description préalable des services. Nous avons réalisé un registre de services Web (WSR) qui se base sur le modèle de représentation de services Web et ainsi facilite ces trois étapes. D'un point de vue technologique, l'implémentation de WSR repose sur le système de représentation de connaissances par objets AROM. L'utilisation d'AROM permet d'intégrer une dimension sémantique au modèle de représentation et d'étendre facilement le modèle à partir de modèles existants formalisés par le biais de ce système.

Composition de services Web. Afin de concevoir des applications à base de services Web, le client doit faire appel à un ensemble de services Web. Afin de faciliter cette étape, des solutions existent afin de définir et d'exécuter la composition de services Web. La particularité de notre travail repose sur le modèle de composition de services Web (ProbCWS). Ce dernier décompose la requête du client en une sélection de services Web de granularité faible. ProbCWS s'inspire des méthodes de résolution de problèmes à base de tâches. Ce modèle est la clé de voûte de la plate-forme de génération d'applications (GenAWS). Nous avons choisi pour implémenter cette plate-forme le langage de résolution de problèmes AROMTasks, sous-jacent au système de représentation de connaissances AROM. Nous proposons une extension de ce langage, AROMTasks-CWS, qui prend en compte les concepts d'adaptation au contexte d'utilisation inhérents à ProbCWS. L'avantage de l'utilisation de ce langage est qu'il met en œuvre la surveillance de l'exécution de la composition et permet ainsi de rendre la composition de services Web évolutive.

La technique d'adaptation …	Adaptation par conversion	Adaptation par déploiement de service d'adaptation	Adaptation par sélection de service	Adaptation par sélection de composition	Adaptation par compensation
… met en œuvre …	- Adaptation du résultat	- Adaptation du résultat - Adaptation de la demande - Adaptation de la composition	- Adaptation de la demande - Adaptation de la composition	- Adaptation de la composition	- Adaptation de la demande - Adaptation de la composition
… est évaluée comme étant de …	- Adaptabilité - Adaptativité	- Adaptation minimale	- Adaptation minimale	- Adaptation minimale	- Adaptativité
… est réalisé au sein de …	- ProbCWS - GenAWS	- ProbCWS - GenAWS	- WSR-Model - ProbCWS - GenAWS - WSR	- ProbCWS - GenAWS	- WSR-Model - ProbCWS - GenAWS

Tableau 12.1 Tableau de synthèse de la mise en œuvre de l'adaptation au contexte d'utilisation au sein de nos propositions.

Mise en œuvre de l'adaptation au contexte d'utilisation. La mise en œuvre de l'adaptation est présente dans l'ensemble de notre proposition. En d'autres termes, nous proposons d'intégrer le concept d'adaptation au contexte d'utilisation à la plupart des étapes du cycle de vie d'une application à base de services. Nous avons déterminé et réalisé dans notre travail trois types d'adaptation : adaptation du résultat du service, de la demande de services, de la composition. Chacun de ces types d'adaptation résulte d'une ou plusieurs technique(s) d'adaptation (adaptation par conversion, par déploiement de service d'adaptation, par sélection d'un service

ou d'une composition de services, par compensation, par déploiement de service d'adaptation) et est évalué (adaptation minimale, adaptabilité, adaptativité). Le Tableau 12.1 résume l'utilisation des techniques d'adaptation afin de mettre en œuvre les types d'adaptation. Nous mettons aussi en relief dans ce tableau dans quelle(s) partie(s) de notre proposition les mises en œuvre de l'adaptation sont intégrées.

12.2 Perspectives

Les thèmes principaux liés à ces travaux de thèse (SOA, services Web et adaptation au contexte) sont encore à ce jour en pleine expansion. Preuve en est, de nouveaux projets rassemblant les problématiques liées au sujet de cette thèse voit le jour, tel que le projet SOA4All[80]. L'objectif de ce projet européen, créé en janvier 2008 et soutenu par l'initiative NESSI[81] (*Networked European Software and Services Initiative*), est de fournir une infrastructure et un cadre de travail complets qui intègrent les quatre avancées techniques complémentaires : les **principes et technologies du Web**, le **Web 2.0**, le **Web sémantique**, et la **gestion du contexte**.

Ceci ouvre de nombreuses perspectives à ce travail de thèse. Nous mettons en relief deux types de perspectives : les perspectives qui visent l'extension de notre travail et celle qui visent l'implication de nos propositions au sein de domaines de l'informatique convergents aux nôtres.

La clé de voûte de notre travail de thèse est le modèle de représentation de services Web. Celui-ci peut être facilement étendu afin de prendre en compte de nouveaux concepts sous-jacents à l'utilisation des services Web. Nous notons deux concepts pour lesquels il serait intéressant de faire évoluer le modèle : la qualité de service et les spécificités liées aux services localisés.

La qualité de service. Les travaux issus du domaine de la qualité de services, tels que [Ran, 2003] et [Kalepu *et al.*, 2003], répertorient un ensemble d'informations à prendre en compte pour définir la qualité d'un service. Ces informations sont : le temps de réponse (en ms), le débit (en requête par minute), la disponibilité (en pourcentage), l'accessibilité (en pourcentage), l'analyse de l'interopérabilité (en pourcentage), et le coût (en cents par invocation). Nous imaginons facilement un nouveau package au modèle de représentation de services qui inclue l'ensemble de ces informations. Ces dernières pourront être, soit renseignées par le fournisseur du service (comme pour le coût du service), soit calculées à la suite de l'invocation du service décrit (pour les autres informations). La description de la qualité de services peut ensuite faciliter la tâche de sélection mise en œuvre dans le registre WSR ou la plate-forme GenAWS.

Les services localisés. Les dernières avancées en termes de réseaux sans fil et de capacités techniques des dispositifs mobiles permettent aux Systèmes d'Information Géographique (SIG) d'être mobiles. Cette nouvelle génération de SIG repose sur des services localisés (LBS – *Location-Based services*) [Frank *et al.*, 2004], services exploitant la localisation de l'utilisateur à travers différents capteurs (tels qu'un GPS, un compas, un lecteur RFID). La mutation des SIG traditionnels (fixes) en SIG mobiles entraîne notamment des difficultés en terme de mémoire nécessaire et de vitesse d'exécution. Afin d'éviter ces types de problèmes, il est possible de décomposer l'application en de multiples entités [Hinze *et al.*, 2003]. Cette décomposition permet également de réutiliser des fonctionnalités pré-existantes. Comme nous l'avons montré dans cette thèse, la réutilisation de services s'appuie principalement sur des descriptions standard. L'*Open Geospatial Consortium* (OGC) propose des spécifications standard afin de décrire des services liés à l'information spatialisée, telles que les spécifications WMS (*Web Map*

[80] http://www.soa4all.org/

[81] http://www.nessi-europe.com/

Service) [OGC, 2004] et WFS (*Web Feature Service*) [OGC, 2005]. Les services de type WMS fournissent des cartes, tandis que les services de type WFS permettent de récupérer et de mettre à jour des données géographiques répondant au formant d'échange GML (*Geography Markup Language*) [OGC, 2007]. L'ensemble de notre proposition peut être un moyen d'utiliser (et de réutiliser) les LBS afin d'implémenter des SIG adaptés au contexte d'utilisation. L'extension du modèle de représentation de services pourra prendre en compte les spécifications de l'OGC telles que WFS et WMS.

D'un point de vue implication de nos travaux de thèse au sein d'autres domaines de l'informatique, nous pensons principalement aux domaines des réseaux de capteurs et de la sécurité des données, nécessaires durant la phase d'exécution.

Les réseaux de capteurs. Notre travail propose un modèle pour représenter les services utilisés afin de concevoir des applications adaptées au contexte d'utilisation. Le contexte pour lequel l'application conçue est adaptée est, dans notre travail, renseigné manuellement par le concepteur. Notre travail de thèse pourra donc facilement s'intégrer dans un projet mêlant les architectures orientées services et les réseaux de capteurs [Wu *et al.,* 2007]. Les réseaux de capteurs permettent de récupérer des données environnementales à travers un ensemble de capteurs tels qu'un accéléromètre, un capteur de luminosité, ou encore un GPS. Nous pouvons imaginer que notre travail soit accessible à des clients mobiles possédant un dispositif équipé de capteurs. Ainsi le contexte d'utilisation auquel l'application est adaptée, sera connu par l'intermédiaire d'un ensemble de capteurs.

La sécurité. Le domaine de la sécurité est important à prendre en compte dans notre travail pour deux raisons. Tout d'abord, le fait que nous implémentons des applications pervasives, de plus basées sur la technologie des services Web, engendre une sensibilité certaine concernant la transmission des données. Ensuite, le fait que nous proposons de l'adaptation au contexte d'utilisation implique la manipulation des données personnelles, par conséquent sensibles. En effet d'après [Langheinrich, 2001], la gestion du contexte doit être réalisée en respectant la vie privée de l'utilisateur. Des travaux de thèse, en cours dans notre équipe, proposent une solution de gestion d'informations contextuelles qui permet le stockage, la distribution, et la consultation de données du contexte, en considérant la sécurité et le respect de la vie privée exigés par les utilisateurs et par les services [Bringel Filho, 2007]. Ces travaux pourraient être intégrés à GenAWS afin d'inclure, lors de l'exécution des applications, la notion de sécurité des données.

Bibliographie

[Adler *et al.*, 2001] Adler, S., Berglund, A., Caruso, J., Deach, S., Graham, T., Grosso, P., Gutentag, E., Milowski, A., Parnell, S., Richman, J., Zilles, S. Extensible Stylesheet Language (XSL) Version 1.0. W3C Recommendation **[en ligne]**, 2001. Disponible sur : <http://www.w3.org/TR/xsl/>.

[Aggarwal *et al.*, 2004] Aggarwal, R., Verma, K., Miller, J., Milnor, W. Constraint Driven Web Service Composition in METEOR-S. **In:** Procs of the IEEE International Conference on Services Computing (SCC'04), Sept. 2004, Shanghai, China. IEEE Computer Society, 2004, pp.23-30.

[Akkiraju *et al.,* 2005] Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M.-T., Sheth A., Verma, K. Web Service Semantics – WSDL-S. W3C Member Submission **[en ligne]**, 2005. Disponible sur : <http://www.w3.org/Submission/WSDL-S>.

[Al-Masri *et al.*, 2007a] Al-Masri, E., Mahmoud, Q.H. Crawling Multiple UDDI Business Registries. **In:** Procs of the 16th International Conference on World Wide Web (WWW2007), May 2007, Banff, Alberta, Canada. ACM, 2007, pp.1255-1256.

[Al-Masri *et al.*, 2007b] Al-Masri, E., Mahmoud, Q.H. Discovering the Best Web Service. **In:** Procs of the 16th International Conference on World Wide Web (WWW2007), May 2007, Banff, Alberta, Canada. ACM, 2007, pp.1257-1258.

[Al-Masri *et al.*, 2007c] Al-Masri, E., Mahmoud, Q.H. QoS-based Discovery and Ranking of Web Services. **In:** Procs of the 16th International Conference on Computer Communications and Networks (ICCCN 2007), Aug. 2007, Honolulu, Hawaii, USA. IEEE, 2007, pp.529-534.

[Alonso *et al.*, 2004] Alonso, G., Casati, F., Kuno, H., Machiraju, V. Web Services: Concepts, Architectures and Applications. Springer, 2004, 354p.

[Andrews *et al.,* 2003] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S. Business Process Execution Language for Web Services, Version 1.1. IBM Specification **[en ligne]**, 2003. Disponible sur : <http://www-128.ibm.com/developerworks/library/specification/ws-bpel>.

[Ankolekar *et al.*, 2002] Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D.L., McDermott, D.V., McIlraith, S.A., Narayanan, S., Paolucci, M., Payne, T.R., Sycara, K.P. DAMLS: Web Service Description for the Semantic Web. **In:** Procs of the International Semantic Web Conference (ISWC'02), June 2002, Sardinia, Italy. LNCS Springer, 2002, pp.348-363.

[Ankolekar *et al.*, 2004] Ankolekar, A., Martin, D., McGuinness, D., McIlraith, S., Paolucci, M., Parsia, B. OWL-S' Relationship to Selected Other Technologies. W3C Member Submission **[en ligne]**, 2004. Disponible sur : <http://www.w3.org/Submission/OWL-S-related>.

[Arkin *et al.,* 2002] Arkin, A., Askary, S., Fordin, S., Jekeli, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S., Takacsi, P., Trickovic, I., Zimek, S. Web Service Choreography Interface (WSCI) 1.0. W3C Note **[en ligne]**, 2002. Disponible sur : <http://www.w3.org/TR/wsci>.

[Austin *et al.,* 2004] Austin, D., Barbir, A., Peters, E., Ross-Talbot, S. Web Services Choreography Requirements. W3C Working Draft **[en ligne]**, 2004. Disponible sur : <http://www.w3.org/TR/ws-chor-reqs>.

[Baader *et al.*, 2003] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F., editors. The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge, UK, 2003, 555p.

[Baader *et al.,* 2005] Baader, F., Lutz, C., Milicic, M., Sattker, U., Wolter, F. A Description Logic Based Approach to Reasoning about Web Services. **In:** Procs of the International Conference WWW, Workshop on Web Service Semantics: Towards Dynamic Business Integration (WSS2005), May 2005, Chiba Japan.

[Bachlechner *et al.*, 2006] Bachlechner, D., Siorpaes, K., Fensel, D., Toma I. Web Service Discovery – A Reality Check. DERI – Digital Enterprise Research Institute. Technical Report **[en ligne]**, 2006. Disponible sur : <http://www.deri.ie/fileadmin/documents/DERI-TR-2006-01-17.pdf>.

[Balke *et al.,* 2003a] Balke, W.-T., Wagner, M. Towards Personalized Selection of Web Services. **In:** Procs of the Alternate Papers Track of the International World Wide Web Conference (WWW 2003), May 2003, Budapest, Hungary. ACM Press.

[Balke *et al.,* 2003b] Balke, W.-T., Wagner, M. Cooperative Discovery for User-centered Web Service Provisioning. **In:** Procs of the International Conference on Web Services (ICWS 2003), June 2003, Las Vegas, USA. CSREA Press, pp.191-197.

[Banavar *et al.*, 2000] Banavar, G., Beck, J., Gluzberg, E., Munson, J., Sussman, J., Zukowski, D. Challenges: an Application Model for Pervasive Computing. **In:** Procs of the 6th International Conference on Mobile Computing and Networking, Boston, Massachusetts, United States, Aug. 2000. ACM Press, 2000, pp.266-274.

[Baresi *et al.*, 2003] Baresi, L., Bianchini, D., De Antonellis, V., Fugini, M.G., Pernici, B., Plebani, P. Context-Aware Composition of E-Services. **In:** Procs of the 4th International Workshop of Technology for E-Services (TES 2003), Sept. 2003, Berlin, Germany. LNCS Springer, 2003, pp.28-41.

[Barros *et al.,* 2005a] Barros, A., Dumas, M., Oaks, P. A Critical Overview of the Web Services Choreography Description Language. Business Process Trends White Paper **[en ligne]**, 2005. Disponible sur : <http://www.bptrends.com/publicationfiles/03-05%20WP%20WS-CDL%20Barros%20et%20al.pdf>.

[Barros *et al.*, 2005b] Barros A., Dumas, M., Oaks, P. Standards for Web Service Choreography and Orchestration: Status and Perspectives. **In:** Procs of the 3rd International Conference the Business Process Management (BPM 2005), 1st International Workshop on Web Service Choreography and Orchestration for Business Process Management, Nancy, France, Sept. 2005, pp.1-15.

[Barth, 2007] Barth, I. Le mariage du frigo et du téléphone portable : Vers une modélisation de l'espace domestique comme enjeu social de la diffusion des TIC. **In :** Actes du 14ème colloque Informatique et Société 2007 : De l'insécurité numérique à la vulnérabilité de la société **[en ligne]**. Atelier n°1 : vie quotidienne. Disponible sur : <http://www.creis.sgdg.org/colloques%20creis/2007/iwan%20Barth.pdf>.

[Bellman, 1978] Bellman, R.E. An introduction to Artificial Intelligence: Can Computers Think? San Francisco: Boyd & Fraser Publishing Company, 1978.

[Ben Mokhtar *et al.*, 2005a] Ben Mokhtar, S., Georgantas, N., Issarny, V. Ad Hoc Composition of User Tasks in Pervasive Computing Environments. **In:** Procs of the 4th Workshop on Software Composition (SC 2005), April 2005, Edinburgh, UK. LNCS Springer, pp.31-46.

[Ben Mokhtar *et al.*, 2005b] Ben Mokhtar, S., Fournier, D., Georgantas, N., Issarny, V. Context-Aware Service Composition in Pervasive Computing Environments. **In:** Procs of the 2nd International Workshop Rapid Integration of Software Engineering Techniques (RISE 2005), Sept. 2005, Heraklion, Crete, Greece. LNCS Springer, pp.129-144.

[Benatallah *et al.*, 2002] Benatallah, B., Dumas, M., Fauvet, M.-C., Rabhi, F.A., Sheng, Q.Z. Overview of Some Patterns for Architecting and Managing Composite Web Services. ACM SIGecom Exchanges, 2002, vol.3, n°3, pp.9-16.

[Benatallah *et al.,* 2003] Benatallah, B., Sheng, Q., Dumas, M. The SELF-SERV environment for Web services composition. IEEE Internet Computing, 2003, vol.7, n°1, pp.40-48.

[Benatallah *et al.,* 2005] Benatallah B., Dijkman R., Dumas M., Maamar Z. Service Composition: Concepts, Techniques, Tools and Trends. **In:** Stojanovic Z., Dahanayake A., Eds. Service-Oriented Software Engineering: Challenges and Practices. Idea Group Inc (IGI), 2005, pp.48-66.

[Benslimane *et al.*, 2007] Benslimane, D., Maamar, Z. Preface – Context-Aware Web Services. Distrib Parallel Databases, 2007, vol.21, pp.1-3.

[Benslimane *et al.*, 2008] Benslimane, D., Dustdar, S., Sheth, A. Services Mashups: The New Generation of Web Applications. IEEE Internet Computing, 2008, vol.12, n°5, pp.13-15.

[Berners-Lee *et al.*, 2001] Berners-Lee, T., Hendler, J., Lassila, O. The Semantic Web. **In:** Scientific Americain, 2001.

[Booch *et al.*, 1998] Booch, G., Rumbaugh, J., Jacobson, I. The Unified Modeling Language User Guide. Addison Wesley, 1998, 512 p.

[Bray *et al.*, 2006] Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F., Cowan, J. Extensible Markup Language (XML) 1.1 (Second Edition). W3C Recommendation **[en ligne]**, 2006. Disponible sur : <http://www.w3.org/TR/xml11>.

[Breuker *et al.*, 1988] Breuker J., Wielinga B. Models of Expertise in Knowledge Acquisition. **In:** Guida G., Tasso C. (Editors). Topics in Expert System Design, North Holland Publishing Company, Amsterdam, 1988, pp.265-295.

[Bringel Filho, 2007] Bringel Filho, J. Gestion d'informations contextuelles relatives à la vie privée dans les systèmes pervasifs. **In:** Actes du XXVème Congrès INFORSID, Perros-Guirec, France, mai 2007.

[Brown *et al.*, 1997] Brown, P.J., Bovey, J.D., Chen, X. Context-Aware Applications: From the Laboratory to the Marketplace. IEEE Personal Communications, 1997, vol.4, n°5, pp.58-64.

[Brown, 1996] Brown, P.J. The Stick-e Document: a Framework for Creating Context-Aware Applications. Electronic Publishing, 1996, vol.8, n°2&3, pp.259-272.

[Bruley *et al.*, 2003] Bruley, C., Genoud, P., Dupierris, V. Guide utilisateur AROM v2.0. INRIA Rhône-Alpes **[en ligne]**, 2003. Disponible sur : <ftp://ftp.inrialpes.fr/pub/romans/logiciels/arom2/ug.pdf>.

[Brusilovsky, 1998] Brusilovsky P. Methods and Techniques of Adaptive Hypermedia. **In:** Brusilovsky P., Kobsa A., Vassileva J. (Eds.), Adaptive Hypertext and Hypermedia. Kluwer Academic Publishers, 1998, pp.1-43.

[Brusilovsky, 2001] Brusilovsky, P. Adaptive Hypermedia. User Modeling and User-Adapted Interaction, vol. 11, n°1-2, 2001, pp.87-110.

[Bucur *et al.*, 2005] Bucur, O., Beaume, P., Boissier, O. Définition et représentation du contexte pour des agents sensibles au contexte. **In:** Actes des deuxièmes journées francophones : Mobilité et Ubiquité 2005 (UbiMob'05), mai-juin 2005, Grenoble, France. pp.13-16.

[Cabral *et al.*, 2006] Cabral, L., Domingue, J., Galizia, S., Gugliotta, A., Tanasescu, V., Pedrinaci, C., Norton, B. IRS-III: A Broker for Semantic Web Services Based Applications. **In:** Procs of the 5th International Semantic Web Conference (ISWC 2006), Nov. 2006, Athens, GA, USA. LNCS Springer, 2006, pp.201-214.

[Carman *et al.*, 2003] Carman, M., Serafini, L., Traverso, P. Web Service Composition as Planning. **In:** Procs of the 13th International Conference on Automated Planning and Scheduling (ICAPS 2003), Workshop on Planning for Web Services, June 2003, Trento, Italy.

[Carrillo Ramos, 2007] Carrillo Ramos, A. Agents ubiquitaires pour un accès adapté aux systèmes d'information : le framework PUMAS **[en ligne]**. Thèse de doctorat en Informatique. Université Joseph Fourier, Grenoble I, 2007, 210p. Disponible sur : <http://hal.archives-ouvertes.fr/docs/00/13/69/31/PDF/These-Carrillo-vd.pdf>.

[Chabalier, 2004] Chabalier, J. Acquisition incrémentale et représentation des systèmes intégrés bactériens par une approche orientée-objet. Thèse de doctorat en bio-Informatique. Université de Provence, Aix-Marseille I, 2004, 136p.

[Chandrasekaran *et al.*, 1988] Chandrasekaran B., Smith J., Sticklen J. Generic Tasks as Building Blocks for Knowledge-Based Systems: the Diagnosis and Routine Design Examples. The Knowledge Engineering Review, 1988, vol.3, n°3, pp.183-210.

[Chandrasekaran, 1990] Chandrasekaran, B. Design Problem Solving: A Task Analysis. AI Magazine, 1990, vol.11, pp.59-71, 1990.

[Chen *et al.*, 2000] Chen, G., Kotz, D. A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Dartmouth, 2000, 16p.

[Chen *et al.*, 2004] Chen, H., Perich, F., Finin, T.W., Joshi, A. SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. **In:** Procs of the 1st Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous 2004), Networking and Services, Aug. 2004, Cambridge, MA, USA. IEEE Computer Society, 2004, pp.258-267.

[Chinnici *et al.*, 2007] Chinnici, R., Moreau, J.-J., Ryman, A., Weerawarana, S. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Recommendation **[en ligne],** 2007. Disponible sur : <http://www.w3.org/TR/wsdl20>.

[Christensen *et al.*, 2001] Christensen, E., Curbera, F., Greg Meredith, G., Weerawarana, S. Web Services Description Language (WSDL) 1.1. W3C Note **[en ligne]**, 2001. Disponible sur : <http://www.w3.org/TR/wsdl>.

[Clark, 1999] Clark, J. XSL Transformations (XSLT) Version 1.0. W3C Recommendation **[en ligne]**, 1999. Disponible sur : <http://www.w3.org/TR/xslt>.

[Claro *et al.*, 2005] Claro D., Albers P., Hao J. Selecting Web Services for Optimal Composition. **In:** Procs of the 3rd International Conference on Web Services (ICWS 2005), 2nd International Workshop on Semantic and Dynamic Web Processes, Orlando, Florida, USA, July 2005, pp.32-45.

[Claro *et al.*, 2006] Claro D., Albers P., Hao J. Web Services Composition. **In:** Cardoso, J., Sheth, A., Eds. Semantic Web Services, Processes and Applications. Springer, 2006, pp.195-225.

[Clement *et al.*, 2004] Clement, L. Hately, A., von Riegen, C., Rogers, T. UDDI v.3.0.2. OASIS Specification **[en ligne]**, 2004.Disponible sur : <http://uddi.org/pubs/uddi_v3.htm>.

[Cohen *et al.*, 1982] Cohen, P.R., Feizenbaum, E.A. The Handbook of Artificial Intelligence. Cohen, P.R., Feizenbaum, E.A. (Editors), vol.3. William Kaufmann, Los Altos, California, 1982.

[Context, 1997] **In:** Procs of the 1st International and Interdisciplinary Conference on Modeling and Using Context (Context-97), Feb. 1997, Rio de Janeiro, Brazil. Federal University of Rio de Janeiro Ed.

[Crampé, 1994] Crampé, I. Sémantique des tâches décomposables et spécialisables. Rapport de DEA d'Informatique. Université Joseph Fourier, Grenoble I, 1994, 89p.

[Crubezy *et al.*, 2003] Crubezy, M., Motta, E., Lu, W., Musen, M. Configuring Online Problem-Solving Resources with the Internet Reasoning Service. IEEE Intelligent Systems, 2003, vol.2, pp.34-42.

[Crusson, 2003] Crusson, T. Business Process Management : de la modélisation à l'exécution - Positionnement par rapport aux Architectures Orientées Services. White paper. Intalio white paper **[en ligne]**, 2003. Disponible sur : <http://www.dotnetguru.org/downloads/BPMWhitepaper.pdf>.

[Cuddy *et al.*, 2005] Cuddy, S., Katchabaw, M., Lutfiyya, H. Context-Aware Service Selection Based on Dynamic and Static Service Attributes. **In:** Procs of the IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, Aug. 2005, Montréal, Canada. pp.148-159.

[Curbera *et al.*, 2002] Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., Weerawarana, S. Unraveling the Web services Web: An introduction to SOAP, WSDL, and UDDI. IEEE Internet Computing, 2002, vol.6, n°2, pp.86-93.

[De Bra *et al.*, 2008] De Bra, P., Aerts, A., Berden, B., De Lange, B., Rousseau, B., Santic, T. Smots, D., Stash, N. AHA ! The Adaptive Hypermedia Architetcure. **In:** Procs of the 14th ACM Conference on Hypertext and Hypermedia (HT 03), Nottingham, United Kingdom, Aug. 2003, pp.81-84.

[Dey *et al.*, 2000] Dey, A.K., Abowd, G.D. Towards a Better Understanding of Context and Context-Awareness. **In:** Procs of the Conference on Human Factors in Computing Systems (CHI'2000), Workshop on the What, Who, Where, When, and How of Context-Awareness, April 2000, The Hague, The Netherlands.

[Dey *et al.*, 2001] Dey, A., Abowd, G., Salber, D. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Human-Computer Interaction, 2001, vol.16, n°2, 3, 4, pp.97-166.

[Dey, 2001] Dey, AK. Understanding and Using Context. Personal and Ubiquitous Computing, 2001, vol.5, n°1, pp.4-7.

[Dietrich *et al.*, 1993] Dieterich, H., Malinowski, U., Khme, T., Schneider-Hufschmidt, M. State of the Art in Adaptive User Interfaces. **In:** Schneider-Hufschmidt, M., Khme, T., Malinowski, U. (Eds.), Adaptive User Interfaces: Principle and Practice. Amsterdam, North Holland, 1993, pp.13-48.

[Domingue *et al.*, 2006] Domingue, J., Galizia, S., Cabral, L. The Choregraphy Model for IRS-III. **In:** Procs of the Hawaii International Conference on System Sciences (HICSS 2006), Jan. 2006, Kauai, Hawaii, USA. IEEE Computer Society 2006, track 3, pp.62-70.

[Dovey *et al.*, 2005] Dovey, M., Kostadinov, I., Giddy, J., Green, P., Berry, D., Chonan, D., Wang, X. UK Engineering Tasks Force Evaluation of UDDI for UK e-Science. UK Technical Report

[en ligne], 2005, 21p. Disponible sur : <http://www.nesc.ac.uk/technical_papers/UKeS-2005-04.pdf>.

[Duarte-Amaya, 2007] Duarte-Amaya, H. Tcows Canevas pour la composition de services web avec propriéetées transactionnelles **[en ligne]**. Thèse de doctorat en Informatique. Université Joseph Fourier, Grenoble I, 2007, 159p. Disponible sur : <http://hal.archives-ouvertes.fr/docs/00/19/29/24/PDF/TheseHelgaDuarteAmaya.pdf>.

[Erl, 2005] Erl, T. Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, 2005, 760p.

[Erol *et al.*, 1994] Erol, K, Hendler, J., Nau, D.S. HTN Planning: Complexity and Expressivity. **In:** Procs of the 12th National Conference on Artificial Intelligence (AAAI 1994), July-Aug. 1994, Seattle, WA, USA. AAAI Press, pp.1123-1128.

[Farrell *et al.,* 2007] Farrell, J., Lausen, H. Semantic Annotations for WSDL and XML Schema. W3C Recommendation **[en ligne]**, 2007. Disponible sur: <http://www.w3.org/TR/sawsdl/>.

[Fensel *et al.*, 2002a] Fensel, D., Bussler, C., Maedche, A. Semantic Web Enabled Web Services. **In:** Procs of the 1st International Semantic Web Conference (ISWC 2002), Sardinia, Italy, June 2002. LNCS Springer 2002, pp.1-2.

[Fensel *et al.*, 2002b] Fensel D., Bussler C., Ding Y., Omelayenko B. The Web Service Modeling Framework WSMF. Electron Commerce Res, vol.1, n°2, 2002, pp.113–137.

[Ferraiolo *et al.*, 2003] Ferraiolo, J., Fujisawa, J., Jackson, D., Scalable Vector Graphics (SVG) 1.1 Specification, W3C Recommendation **[en ligne]**, 2003, Disponible sur : <http://www.w3.org/TR/SVG11/>.

[Ferreira da Cunha, 1999] Ferreira da Cunha, S. Comparaison de résolutions de problèmes exprimées par un modèle de tâches. Thèse de doctorat en Informatique. Université Joseph Fourier, Grenoble I, 1999, 130p.

[Ferris *et al.*, 2007] Ferris, C., Karmarkar, A., Yendluri, P. Basic Profile Version 2.0. WS-I Working Group Draft **[en ligne]**, 2007. Disponible sur : <http://www.ws-i.org/Profiles/BasicProfile-2.0.html>.

[Fikes *et al.*, 2003] Fikes, R., Hayes, P., Horrocks, I. OWL-QL – A Language for Deductive Query Answering on the Semantic Web. Technical Report KSL-03-14 **[en ligne]**, Knowledge Systems Laboratory, Stanford University, CA, 2003, 11p. Disponible sur : http://ksl.stanford.edu/KSL_Abstracts/KSL-03-14.html.

[Frank *et al.*, 2004] Frank, C., Caduff, D., Wuersch, M. From GIS to LBS, An Intelligent Mobile GIS. IfGI prints, Münster, vol.22, 2004.

[Frasincar *et al.*, 2002] Frasincar, F., Houben, G.J. Hypermedia Presentation Adaptation on the Semantic Web. **In:** Procs of the 2nd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH 2002), May 2002, Malaga, Spain. LNCS Springer 2002, pp.133-142.

[Garlan *et al.*, 2002] Garlan, D., Siewiorek, D.P., Smailagic, A., Steenkiste, P. Project Aura: Toward Distraction-Free Pervasive Computing. IEEE Pervasive Computing, 2002, vol.1, n°2, pp.22-31.

[Genoud *et al.*, 2005] Genoud, P., Bruley, C., Ziébelin, D. Guide Utilisateur AROMTasks. INRIA Rhône-Alpes, 2005.

[Gómez Pérez *et al.*, 1999] Gómez Pérez A., Benjamins, V.R. Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem-Solving Methods. **In:** Procs of the 6th International Joint Conference on Artificial Intelligence (IJCAI 1999), Wokshop on Ontologies and Problem-Solving Methods (KRR5), Stockholm, Sweden, Aug. 1999. CEUR Workshop Proceedings, 1999, 15p.

[Gong, 2005] Gong, L. Contextual Modeling and Applications. **In:** Procs of IEEE International Conference on Systems, Man, and Cybernetics (SMC) 2005, vol.1, Oct. 2005, Waikoloa, Hawaii, USA, 2005, pp.381-386.

[Gruber, 1993] Gruber, T.R. A Translation Approach to Portable Ontologies. Knowledge Acquisition, 1993, vol.5, n°2, pp.199-220.

[Guedalia *et al.*¸ 1999] Guedalia, I.D., London, M., Werman, M. An On-Line Agglomerative Clustering Method for Nonstationary Data. Neural Computer, 1999, vol.11, pp.521–540.

[Guttman *et al.*, 1999] Guttman, E., Perkins, C., Veizades, J., Day, M. Service Location Protocol, Version 2. RFC 2608 **[en ligne]**, 1999. Disponible sur : <http://www.rfc-editor.org/rfc/rfc2608.txt>.

[Haas *et al.,* 2004] Haas, H., Brown, A. Web Services Glossary. W3C Working Group Note **[en ligne]**, 2004. Disponible sur: <http://www.w3.org/TR/ws-gloss/>.

[Haton *et al.,* 1991] Haton J.P., Bouzid, N., Charpillet, F., Haton, M.C., Lâasri, B., Lâasri, H., Marquis, P., Mondot, T., Napoli, A., Pitrat, J. Le raisonnement en intelligence artificielle : modèles, techniques et architectures pour les systèmes à bases de connaissances. InterEditions, Paris, 1991, 480p.

[Hendler, 2008] Hendler, J.Web 3.0: Chicken Farms on the Semantic Web. Computer, 2008, vol. 41, n°1, pp.106-108.

[Heß *et al.*¸ 2004] Heß, A., Johnston, E., Kushmerick, N. ASSAM: A Tool for Semi-automatically Annotating Semantic Web Services. **In:** Procs of the 3rd International Semantic Web Conference (ISWC2004), Nov. 2004, Hiroshima, Japan. LNCS Springer, 2004, pp.320-334.

[Hinze *et al.*, 2003] Hinze, A., Voissard, A. Location- and Time-Based Information Delivery in Tourism. **In:** Procs of the Advances in Spatial and Temporal Databases (SSTD 2003), July 2003, Santorini Island, Greece. LNCS Springer, pp.489-507.

[Hobbs *et al.*, 2006] Hobbs, J.R., Pan, F. Time Ontology in OWL. W3C Working Draft **[en ligne]**, 2006. Disponible sur : <http://www.w3.org/TR/owl-time/>.

[Horn *et al.,* 1991] Horn, W. The challenge of deep models, inference structures and abstracts tasks. Applied Artificial Intelligence, 1991, vol.5, pp.87-98.

[Horrocks *et al.*, 2004] Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission **[en ligne]**, 2004. Disponible sur : <http://www.w3.org/Submission/SWRL/>.

[Jang *et al.*, 2003] Jang, S., Who, W. Ubi-UCAM: A Unified Context-Aware Application Model. **In:** Procs of the 4th International and Interdisciplinary Conference on Modeling and Using Context (Context-97), June 2003, Stanford, CA, USA. LNAI Springer, 2003, pp.178-189.

[Jones, 2005] Steve, J. Toward an Acceptable Definition of Service. IEEE Software, 2005, vol.22, n°3, pp.87-93.

[Kalepu *et al.*, 2003] Kalepu, S., Krishnaswamy, S., Loke, S.W. Verity: A QoS Metric for Selecting Web Services and Providers. **In:** Procs of the 4th International Conference on Web Information Systems Engineering Workshops (WISEW'03), Dec. 2003, Roma, Italy. pp.131-139.

[Kang *et al.*, 2007a] Kang, Z., Wang, H., Hung, P.C.K. WS-CDL+: An Extended WS-CDL Execution Engine for Web Service Collaboration. **In:** Procs of the IEEE International Conference on Web Services (ICWS 2007), July 2007, Salt Lake City, Utah, USA. IEEE Computer Society 2007, pp.928-935.

[Kang *et al.*, 2007b] Kang, Z., Wang, H., Hung, P.C.K. WS-CDL+ for web service collaboration. Information Systems Frontiers, 2007, vol.9, n°4, pp.375-389.

[Kappel *et al.*, 2000] Kappel, G., Retschitzegger, W., Schwinger, W. Modeling Customizable Web Applications – A Requirement's Perspective. **In:** Procs of the International Conference on Digital Libraries (ICDL 2000), Nov. 2000, Kyoto, Japan.

[Kavantzas *et al.,* 2005] Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y., Barreto, C. Web Services Choreography Description Language Version 1.0. W3C Candidate Recommendation **[en ligne]**, 2005. Disponible sur : <http://www.w3.org/TR/ws-cdl-10>.

[Keidl *et al.,* 2004] Keidl, M., Kemper, A. Towards Context-Aware Adaptable Web Services. **In:** Procs of the 13th international World Wide Web Conference (WWW 2004), May 2004, NewYork, USA, pp.55-65.

[Kirsch Pinheiro, 2006] Kirsch Pinheiro, M. Adaptation Contextuelle et Personnalisée de l'Information de Conscience de Groupe au sein des Systèmes d'Information Coopératifs **[en ligne]**. Thèse de doctorat en Informatique. Université Joseph Fourier, Grenoble I, 2006, 272p. Disponible sur : <http://hal.archives-ouvertes.fr/docs/00/10/84/95/PDF/TheseManueleKirschPinheiro-Final.pdf>.

[Kiss, 2007] Kiss, C. Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 2.0. W3C Working Draft **[en ligne]**, 2007. Disponible sur : <http://www.w3.org/TR/CCPP-struct-vocab2/>.

[Klyne *et al.*, 2004a] Klyne, G., Reynolds, F., Woodrow, C., Ohto, H., Hjelm, J., Butler, M.-H., Tran, L. Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0. W3C Recommendation **[en ligne]**, 2004. Disponible sur : <http://www.w3.org/TR/CCPP-struct-vocab/>.

[Klyne *et al.*, 2004b] Klyne, G., Carroll, J.J. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation **[en ligne]**, 2004. Disponible sur : <http://www.w3.org/TR/rdf-concepts/>.

[Knublauch *et al.*, 2004] Knublauch, H., Fergerson, R.W., Noy, N.F., Musen, M.A. The Protégé OWL Plug: An Open Development Environment for Semantic Web Applications. **In:** Procs of the 3rd International Semantic Web Conference, Nov. 2004, Hiroshima, Japan. LNCS Springer, pp.229-246.

[Kobsa *et al.*, 2001] Kobsa, A., Koenemann, J., Pohl, W. Personalized Hypermedia Presentation Techniques for Improving Online Customer Relationships. The Knowledge Engineering Review, 2001, vol.16, n°2, pp.111-155.

[Langheinrich, 2001] Langheinrich, M. Privacy by Design - Principles of Privacy-Aware Ubiquitous Systems. **In:** Procs of the 3rd International Conference of Ubiquitous Computing (Ubicomp 2001), Oct. 2001, Atlanta, Georgia, USA. LNCS Springer, 2001, pp.273-291.

[Lemlouma, 2004] Lemlouma, T. Architecture de négociation et d'adaptation de Services Multimedia dans des Environnements Hétérogènes **[en ligne]**. Thèse de Doctorat. Institut National Polytechnique de Grenoble, 2004, 235p. Disponible sur : <http://tel.archives-ouvertes.fr/docs/00/04/69/15/PDF/tel-00006253.pdf>.

[Leymann, 2001] Leymann F. Web Service Flow Language 1.0. IBM Report **[en ligne]**, 2001. Disponible sur : <http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>.

[Lopez-Velasco *et al.*, 2007a] Lopez-Velasco, C., Villanova-Oliver, M., Gensel, J., Martin, H. Registre de services Web pour le développement d'applications. **In:** Actes du XXVème Congrès INFORSID, Perros-Guirec, France, mai 2007, pp.521-536.

[Lopez-Velasco *et al.*, 2007b] Lopez-Velasco, C., Villanova-Oliver, M., Gensel, J., Martin, H. Mobidic : plate-forme de services réutilisables pour l'implémentation de SIG mobiles adaptés au contexte d'utilisation – La mise en œuvre de la recherche de services de base. **In:** Actes de la conférence Québéco-Française de Développement de la Géomatique (CQFD-Géo 2007), juin 2007.

[Lopez-Velasco, 2005] Lopez-Velasco, C. AWSDL : une extension de WSDL pour des services Web adaptés. **In:** Actes du congrès INFORSID, Grenoble, France, mai 2005, pp.133-148.

[Maness, 2006] Maness, J.M. Library 2.0 Theory: Web 2.0 and Its Implications for Libraries. Webology **[en ligne]**, 2006, vol.3, n°2. Disponible sur : <http://www.webology.ir/2006/v3n2/a25.html>.

[Marcos *et al.*, 2003] Marcos, E., de Castro, V., Vela, B. Representing Web Services with UML: A Case Study. **In:** Procs of the 1st International Conference of Service-Oriented Computing (ICSOC 2003), Trento, Italy, Dec. 2003. LNCS Springer, 2003, pp.17-27.

[Martin *et al.*, 2004] Martin, D., Burstein, M., Hobbs, J., Paolucci, Lassila, O., McDermott, D., McIlraith, S., McGuinness, D., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K. OWL-S: Semantic Markup for Web Services. W3C Member Submission **[en ligne]**, 2004. Disponible sur : <http://www.w3.org/Submission/OWL-S>.

[McDermott, 2002] McDermott, D. Estimated-Regression Planning for Interactions with Web Services. **In:** Procs of the 6th International Conference on Artificial Intelligence Planning Systems, April 2002, Toulouse, France. AAAI 2002, pp.201-211.

[McGuiness *et al.*, 2004] McGuiness, D.L., van Harmelen, F. OWL Web Ontology Language Overview. W3C Recommendation **[en ligne]**, 2004. Disponible sur : <http://www.w3.org/TR/owl-features/>.

[McIlraith *et al.*, 2003] McIlraith, S., Martin D. Bringing Semantics to Web Services. IEEE Intelligent Systems. 2003, vol.18, n°1, pp.90-93.

[MEIFT, 2006] Ministère de l'Economie, de l'Industrie et des Finances. Technologies clés 2010. Étude du Ministère de l'Economie, de l'Industrie et des Finances **[en ligne]**. Collection Textes clés, Les Éditions de l'Industrie, Paris 2006, 345p. Disponible sur : <http://www.tc-2010.fr>.

[Milner, 1999] Milner, R. Communicating and Mobile Systems: The π-Calculus. Cambridge University Press, Cambridge, UK, 1999, 160p.

[Miron *et al.*, 2007] Miron, A.D., Capponi, C., Gensel, J., Villanova-Oliver, M., Ziébelin, D., and Genoud, P., Rapprocher AROM de OWL. LMO'2007 Langages et Modèles à Objets, (Toulouse, France, march, 2007), 99-116, 2007.

[Mitra *et al.*, 2007] Mitra, N., Lafon, Y. SOAP Version 1.2 Part 0: Primer (Second Edition). W3C Recommendation **[en ligne]**, 2007. Disponible sur : <http://www.w3.org/TR/soap12-part0/>.

[Moisuc, 2007] Moisuc, B. Conception et Mise en Œuvre de Systèmes d'Information Spatio-Temporelle Adaptatifs : le framework ASTIS. Thèse de doctorat en Informatique. Université Joseph Fourier, Grenoble I, 2007, 168p.

[Motta *et al.*, 1999] Motta, E., Fensel, D., Gaspari, M., Benjamins, R. Specifications of Knowledge Components for Reuse. **In:** Procs of the 11th International Conference on Software Engineering and Knowledge Engineering (SEKE 1999), Kaiserslautern, Germany, June 1999. KSI Press, pp.36-43.

[Motta *et al.*, 2003] Motta, E., Domingue, J., Cabral, L., Gaspari, M. IRS-II: A Framework and Infrastructure for Semantic Web Services. **In:** Procs of the 2nd International Semantic Web Conference, Oct. 2003, Sanibel Island, Florida, USA. LNCS Springer, 2003, pp.306-318.

[Motta, 1999] Motta, E. Knowledge Modelling in OCML. **In:** Motta, E., Editor. Reusable Components for Knowledge Modelling – Case Studies in Parametric Design Problem Solving. IOSPress, 1999, pp.47-79.

[Mrissa, 2007] Mrissa, M. Médiation Sémantique Orientée contexte pour la Composition de Services Web. Thèse de doctorat en Informatique. Université Claude Bernard, Lyon I, 2007 ; 113p.

[Nau *et al.*, 2001] Nau, D., Cao, Y., Lotem, A., Munoz-Avila, H. The SHOP Planning System. AI Magazine, 2001, vol.22, n°3, pp.91-94.

[Nau *et al.*, 2003] Nau, D., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F. SHOP2: An HTN Planning System. Journal of Artificial Intelligence Research, 2003, vol.20, pp.379-404.

[Neville *et al.*, 2003] Neville, J., Rattigan, M., Jensen, D. Statistical Relational Learning: Four Claims and a Survey. **In:** Procs of the International Joint Conference on Artificial Intelligence (IJCAI 2003), Workshop on Learning Statistical Models from Relational Data, Aug. 2003, Acapulco, Mexico.

[Nickul *et al.*, 2005] Nickull, D., Connor, M, MacKenzie, C.M., Watson, B., Cowan, M. Service Oriented Architecture and Specialized Messaging Patterns. Adobe Systems. White Paper **[en ligne]**, 2005. Disponible sur : <http://www.adobe.com/enterprise/pdfs/Services_Oriented_Architecture_from_Adobe.pdf>.

[OGC, 2004] Open Geospatial Consortium. OGC Web Map Service Interface. OpenGIS Implementation Specification **[en ligne]**, 2004. Disponible sur : <http://www.opengeospatial.org/standards/wms>.

[OGC, 2005] Open Geospatial Consortium. Web Feature Service Implementation Specification. OpenGIS Implementation Specification **[en ligne]**, 2005. Disponible sur : <http://www.opengeospatial.org/standards/wfs>.

[OGC, 2007] Open Geospation Consortium. OpenGIS Geography Markup Language (GML) Encoding Standard. OpenGIS Specification **[en ligne]**, 2007. Disponible sur : <http://www.opengeospatial.org/standards/gml>.

[Oldham *et al.*, 2004] Oldham, N., Thomas, C., Sheth, A., Verna, K. METEOR-S Web Service Annotation Framework with Machine Learning Classification. **In:** Procs of the 1st International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004 (SWSWPC 2004), July 2004, San Diego, CA, USA. LNCS Springer, 2004, pp.137-146.

[OMA, 2006] OMA – Open Mobile Alliance. White Paper on UAProf Best Practices Guide. Open Mobile Alliance White Paper **[en ligne]**, 2006. Disponible sur : <http://cms.openmobilealliance.org/technical/release_program/docs/UAProf/UAProfBestPracticesGuide-V1_0-20070515-C/OMA-WP-UAProf_Best_Practices_Guide-20060718-C.pdf>.

[Omelayenko *et al.*, 2003] Omelayenko, B., Crubezy, M., Fensel, D., Benjamins, R., Wielinga, B., Motta, E., Musen, M., Ding, Y. UPML: The language and Tool Support for Making the Semantic Web Alive. **In:** Fensel, D., Hendler, J., Wahlster, W., Coord. Spinning the Semantic Web: Bringing the WWW to Its Full Potential. MIT Press, 2003, pp.141-170.

[Oppermann, 1994] Oppermann, R. Introduction. **In:** Oppermann, R. (Ed), Adaptive User Support. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1994, pp.1-13.

[O'Reilly, 2005] O'Reilly T. What is Web 2.0? Design patterns and business models for the next generation of software. O'Reilly Media **[en ligne]**. 2005. Disponible sur : <http://www.oreilly.com/go/web2>.

[Page *et al.*, 2000] Page, M., Gensel, J., Capponi, C., Bruley, C., Genoud, P., Ziébelin, D. Représentation de connaissances au moyen de classes et d'associations : le système AROM. **In:** Actes des 6èmes journées Langages et Modèles à Objets (LMO'2000), Mont Saint-Hilaire, Québec, janv. 2000. Hermès, 2000, pp.91-106.

[Page *et al.*, 2001] Page, M., Gensel, J., Capponi, C., Bruley, C., Genoud, P., Ziébelin, D., Bardou, D., Dupierris, V. A New Approach in Object-Based Knowledge Representation: The AROM System. **In:** Procs of the Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE, 2001), Budapest, Hungary, June 2001. LNAI Springer, 2001, pp.113-118.

[Pan, 2005] Pan, F. A Temporal Aggregates Ontology in OWL for the Semantic Web. **In:** Procs of the American Association for Artificial Intelligence Fall Symposium on Agents and the Semantic Web (AAAI 2005), Arlington, Virginia, USA. AAAI Press, 2005, pp.30-37.

[Paolucci *et al.*, 2003] Paolucci, M., Srinivasan, N. Sycara, K., Nishimura, T. Toward a Semantic Choreography of Web Services: From WSDL to DAML-S. **In:** Procs of the International Conference of Web Services (ICWS03), June 2003, Las Vegas, Nevada, USA. CSREA Press 2003, pp.22-26.

[Papazoglou, 2003] Papazoglou, M. P. Service-Oriented Computing: Concepts, Characteristics and Directions. **In:** Procs of the 4th International Conference on Web Information Systems Engineering (WISE 03), Dec. 2003, Washington, DC, USA, 2003. IEEE Computer Society, 2003, pp.3-12.

[Pascoe, 1998] Pascoe J. Adding Generic Contextual Capabilities to Wearable Computers. **In:** Procs of the 2nd International Symposium on Wearable Computers (ISWC 1998), Oct. 1998, Pittsburgh, Pennsylvania, USA. IEEE Computer Society, 1998, pp.92-99.

[Pashtan *et al.*, 2004] Pashtan, A., Heusser, A., Scheuermann, P. Personal Service Areas for Mobile Web Applications. IEEE Internet Computing, 2004, vol.8, n°6, pp.34-39.

[Payne *et al.*, 2004] Payne T.R., Lassila O. Semantic Web Services. IEEE Intelligent Systems. 2004, vol.19, n°4, pp.14-15.

[Peer, 2005] Peer, J. Web Service Composition as AI Planning – a Survey. Technical Report, University of St. Gallen, Switzerland, 2005, 63p.

[Peltz, 2003] Peltz, C. Web Services Orchestration and Choreography. IEEE Computer, 2003, vol.36, n°10, pp.46-52.

[Power *et al.*, 2004] Power, R., Lewis, D., O'Sullivan, D., Conlan, O., Wade, V. A Context Information Service Using Ontology-Based Queries. **In:** Procs of the 6th International Conference on Ubiquitous Computing (Ubicomp'04), Workshop on Advanced Context Modelling, Reasoning and Management, Sept. 2004, Nottingham, England.

[Preist, 2007] Preist, C. Goals and Vision, Combining Web Services with Semantic Web Technology. **In:** Studer, R., Grimm, S., Abecker A., Coord. Semantic Web Services – Concepts, Technologies, and Applications. Springer Berlin Heidelberg, 2007, pp.159-178.

[Prud'hommeaux *et al.*, 2008] Prud'hommeaux, E, Seaborne, A. SPARQL Query Language for RDF. W3C Recommendation **[en ligne]**, 2008. Disponible sur : <http://www.w3.org/TR/rdf-sparql-query/>.

[Qiu *et al.*, 2006] Qiu, L., Shi, Z., Lin, F. Context Optimization of AI planning for Services Composition. **In:** Procs of the IEEE International Conference on e-Business Engineering (ICEBE 2006), Oct. 2006, Shangai, China.

[Qiu *et al.*, 2007] Qiu, L., Chang, L., Lin, F., Shi, Z. Context Optimization of AI Planning for Semantic Web Services Composition. Service Oriented Computing and Applications, 2007, vol.1, n°2, pp.117-128.

[Ran, 2003] Ran, S. A Model for Web Services Discovery With QoS. ACM SIGecom Exchanges, 2003, vol.4, n°1, pp.1-10.

[Rey *et al.*, 2004] Rey, G., Coutaz, J. Le contexteur : capture et distribution dynamique d'information contextuelle. **In:** Mobilité & Ubiquité'04 (UbiMob'04), Nice, France, 2004, pp.131-138.

[Roman *et al.*, 2005] Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D. Web Service Modeling Ontology. Applied ontology, 2005, vol.1, pp.77-106.

[Rudolph, 2001] Rudolph, L. Project Oxygen: Pervasive, Human-Centric Computing – An Initial Experience. **In:** Procs of the 13th International Conference of Advanced Information Systems Engineering (CAISE 2001), Interlaken, Switzerland, June 2001. LNCS Springer 2001, pp.1-12.

[Russell *et al.*, 1995] Russell, S., Norvig, P. Artificial Intelligence: A Modern Approach. Prentice-Hall, 1995.

[Sacerdoti, 1974] Sacerdoti, E. Planning in a Hierarchy of Abstraction Spaces. Artificial Intelligence, 1974, vol.5, pp.115-135.

[Saha *et al.*, 2003] Saha, D., Mukherjee, A Pervasive Computing: a Paradigm for the 21st Century. Computer, 2003, vol.36, n°3, pp.25-31.

[Sam *et al.*, 2007] Sam, Y., Boucelma, O. Customizable Web Services Description, Discovery and Composition: An Attribute Based Formalisme. **In:** Procs of the 6th International Conference on Web Intelligence, Silicon Valley, CA, USA, Nov. 2007. IEEE Computer Society, pp.143-146.

[Schilit *et al.*, 1994] Schilit, B.N., Adams, N.I., Want, R. Context-Aware Computing Applications. **In:** Procs of the IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'94), Dec. 1994, Santa Cruz, California, USA. IEEE Press, 1994, pp.85-90.

[Schmidt-Schauβ *et al.*, 1989] Schmidt-Schauβ M., Smoka, G. Attributive Concept Descriptions with Complements. Articial Intelligence, 1991, vol.48, n°1, pp.1-26.

[Schreiber *et al.*, 1994] Schreiber, G., Wielinga, B., de Hoog, R., Akkermans, H., van de Velde, W. CommonKADS: A Comprehensive Methodology for KBS Development. IEEE Expert, 1994, vol.9, n°6, pp.28-37.

[Seaborne, 2004] Seaborne, A. RDQL – A Query Language for RDF. W3C Member Submission **[en ligne]**, 2004. Disponible sur : <http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/>.

[Sheng *et al.*, 2002] Sheng, Q.Z., Benatallah, B., Dumas, M., Mak, E.O.-Y. SELF-SERV: A Platform for Rapid Composition of Web Services in a Peer-to-Peer Environment. **In:** Procs of the 28th International Conference on Very Large Data Bases (VLDB 2002), Aug. 2002, Hong Kong, China. Morgan Kaufmann 2002, pp.1051-1054.

[Sheth *et al.*, 2003] Sheth, A.P., Miller, J.A. Web Services: Technical Evolution yet Practical Revolution? IEEE Intelligent Systems (IEEEIS), 2003, vol.18, n°1, pp.78-80.

[Shizuka *et al.*, 2004] Shizuka, M., Ma, J., Lee, J., Miyoshi, Y., Takata, K. A P2P Ubiquitous System for Testing Network Programs. **In:** Procs of the International Conference of the Embedded and Ubiquitous Computing (EUC 2004), Aug. 2004, Aizu-Wakamatsu City, Japan. LNCS Springer 2004, pp.1004-1013.

[Sirin *et al.*, 2004] Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D. HTN Planning for Web Service Composition Using SHOP2. Journal of Web Semantics, 2004, vol.1, n°4, pp.377-396.

[Sivashanmugam *et al.*, 2003] Sivashanmugam, K., Verma, K., Sheth, A., Miller, J. Adding Semantics to Web Services Standards. **In:** Procs of the International Conference on Web Services ICWS03, June, 2003, Las Vegas, Nevada, USA. CSREA Press 2003, pp.395-401.

[Smith, 2007] Smith, K. Device Independent Authoring Language (DIAL). W3C Working Draft **[en ligne]**, 2007. Disponible sur : <http://www.w3.org/TR/dial/>.

[Soukkarieh *et al.*, 2008] Soukkarieh, B., Sédes, F. A Context-Aware Web Information System Based on Web Services. **In:** Procs of the 2nd International Conference on Research Challenge in Information Science (RCIS 2008), Marrakech, Marocco, June 2008.

[Sowa, 1991] Sowa J.F. Principles of Semantic Networks. Sowa J.F. Eds., Morgan Kaufman, 1991.

[Srivastava *et al.*, 2003] Srivastava, B., Koehler, J. Web Service Composition – Current Solutions and Open Problems. **In:** Procs of the 13th International Conference on Automated Planning and Scheduling (ICAPS 2003), Workshop on Planning for Web Services, June 2003, Trento, Italy.

[Stephanidis *et al.*, 1998] Stephanidis, C., Paramythis, A., Sfyrakis, M., Stergiou, A., Maou, N., Leventis, A., Paparoulis, G., Karagiandidis, C. Adaptable and Adaptive User Interfaces for Disabled Users. **In:** Procs of the 5th International Conference on Intelligence in Services and Networks (IS&N98), Technology for Ubiquitous Telecom Services, May 1998, Antwerp, Belgium. LNCS Springer 1998, pp.153–166.

[Storey *et al.*, 2002] Storey, M.A. D., Noy, N.F., Musen, M.A., Best, C., Fergerson, R.W., Ernst, N. Jambalaya: an Interactive Environment for Exploring Ontologies. **In:** Procs of the International Conference on Intelligent User Interfaces, Jan. 2002, San Francisco, California, USA, ACM, 2002, pp.239-239.

[Suraci *et al.*, 2007] Suraci, V., Mignanti, S., Aiuto, A. Context-aware Semantic Service Discovery. **In:** Procs of the 16th IST Mobile and Wireless Communications Summit, July 2007, Budapest, Hungary.

[Sure *et al.*, 2002] Sure, Y., Erdmann, M., Angele, J., Staab, S., Studer, R., Wenke, D. OntoEdit: Collaborative Ontology Development for the Semantic Web. **In:** Procs of the International Semantic Web Conference (ISWC2002), June 2002, Sardinia, Italy. LNCS Springer, 2002, pp.221-235.

[Taher *et al.*, 2008] Taher, T.Y., Fauvet, M.C., Dumas, M., Benslimane, D. Using CEP Technology to Adapt Messages Exchanged by Web Services. **In:** Procs of the 17th International Conference of the World Wide Web (WWW'08), Beijing, China, April 2008. ACM 2008, pp.1231-1232.

[Thatte, 2001] Thatte, S., XLANG: Web Services for Business Process Design. Microsoft Specification, 2001.

[Thompson *et al.*, 2004] Thompson, H.S., Beech, D., Maloney, M., Mendelsohn, N. XML Schema Part 1: Structures Second Edition. W3C Recommendation **[en ligne]**, 2004. Disponible sur : <http://www.w3.org/TR/xmlschema-1/>.

[Tigli *et al.*, 2006] Tigli, J.-Y., Lavirotte, S. Adaptation dynamique à l'environnement d'exécution : un enjeu pur l'informatique mobile et ambiante **[Talk]**. Séminaire, Grenoble, nov., 2006.

[Tsai *et al.*, 2002] Tsai, W.T., Paul, R., Wang, Y., Fan, C., Wang, D. Extending WSDL to Facilitate Web Services Testing. **In:** Procs of the 7th International Symposium on High-Assurance Systems Engineering (HASE 2002), Oct. 2002, Tokyo, Japan. IEEE Computer Society, 2002, pp.171-172.

[Villanova-Oliver, 2002] Villanova-Oliver, M. Adaptabilité dans les systèmes d'information sur le Web : Modélisation et mise en œuvre de l'accès progressif **[en ligne]**. Thèse de doctorat en Informatique. Institut National Polytechnique de Grenoble, 2002, 216p. Disponible sur : <http://tel.archives-ouvertes.fr/docs/00/04/70/61/PDF/tel-00006759.pdf>.

[Vukovic *et al.*, 2004] Vukovic, M., Robinson, P. Adaptive, Planning based, Web Service Composition for context awareness. **In:** Procs of the 2nd International Conference on Pervasive Computing (Pervasive 2004), April 2004, Vienne, Austria.

[Wagner *et al.*, 2002] Wagner, M., Balke, W.-T., Hirschfeld R., Kellerer, W. A Roadmap to Advanced Personalization of Mobile Services. **In:** Procs of the International Conference DOA/ODBASE/CoopIS (Industry Program), 2002, Irvine, CA, USA.

[Wahlster *et al.*, 2006] Wahlster, W., Dengel, A. Web 3.0: Convergence of Web 2.0 and the Semantic Web. Technology Radar Feature Paper, Edition II/2006, Deutsche Telekom Laboratories, pp.1-23.

[Wang *et al.*, 2004] Wang, X., Zhang, D., Gu, T., Pung, H.K. Ontology Based Context Modeling and Reasoning using OWL. **In:** Procs of the 2nd IEEE Conference on Pervasive Computing and Communications Workshops (PerCom 2004 Workshops), March 2004, Orlando, FL, USA. IEEE Computer Society 2004, pp.18-22.

[Weiser, 1991] Weiser, M. The Computer for the 21st Century. Scientific American. 1991, vol.265, n°3, pp.94-104.

[Wiederhold, 1992] Wiederhold, G. Mediators in the Architecture of Future Information Systems. IEEE Computers, 1992, vol.25, n°3, pp.38-49.

[Wielenga *et al.*, 1986] Wielenga B. J., Breuker J. A. Models of Expertise. **In:** Procs of the 7th European Conference on Artificial Intelligence (ECAI 1986), Brighton, United Kingdom, July 1986. North-Holland, 1986, pp.497-509.

[Willamowski, 1994] Willamowski, J. Modélisation de tâches pour la résolution de problèmes en coopération système-utilisateur **[en ligne]**. Thèse de doctorat en Informatique. Université Joseph Fourier, Grenoble I, 1994, 118p. Disponible sur : <http://tel.ccsd.cnrs.fr/documents/archives0/00/00/51/14/tel-00005114-00/tel-00005114.pdf>.

[Winograd, 2001] Winograd T. Architectures for context. Human-Computer Interaction, 2001, pp.402-419.

[Winslett, 1988] Winslett, M. Reasoning About Actions Using a Possible Models Approach. **In:** Procs of the 7th National Conference on Artificial Intelligence (AAAI 1988), Aug. 1988, Saint Paul, Minnesota, USA. AAAI Press / The MIT Press, pp.89-93.

[WMC, 1999] Workflow Management Coalition, Terminology and Glossary. The Workflow Management Coalition Specification **[en ligne]**, 1999. Disponible sur : <http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf>.

[Wohed *et al.*, 2003] Wohed, P., van der Aalst, W., Dumas, M., ter Hofstede, A. Analysis of Web Services Composition Languages: The case of BPEL4WS. **In:** Procs of the 22nd International Conference on Conceptual Modeling, Oct. 2003, Chicago, IL, USA. LNCS Springer 2003, pp.200-215.

[Wu *et al.*, 2007] Wu S.-L., Tseng, Y.-C. Wireless Ad Hoc Networking: Personal-Area, Local-Area, and the Sensory-Area Networks. CRC Press, Auerbach Publications, 664p., 2007.

[Wynen, 2003] Wynen F. Conception et développement d'une plate-forme de coopération inter-organisationnelle : le point de synchronisation. Mémoire d'ingénieur CNAM – Conservatoire Nationale des Arts et Métiers, Nancy, 2003, 102p.

[Yang *et al.*, 2004] Yang, J., Papazoglou, M.P. Service Component for Managing Service Composition Life-Cycle. Information Systems, vol.29, n°2, pp.97-125, 2004.

[Yu *et al.*, 2005] Yu, P., Ma, X., Lu, J. Dynamic Software Architecture Oriented Service Composition and Evolution. **In:** Procs of the 5th International Conference on Computer and Information Technology (CIT 2005), Shanghai, China, Sept. 2005. IEEE Computer Society, 2005, pp.1123-1129.

[Zhu *et al.*, 2005] Zhu, F., Mutka, M.W., Ni, L.M. Service Discovery in Pervasive Computing Environments. Pervasive Computing, 2005, vol.4, n°4, pp.81-90.

Annexe 1 : OntoWS – instanciation du modèle de représentation de services dans le contexte du Web sémantique

Onto-WS, ensemble d'ontologie instanciant WSR-Model, permet d'intégrer ce modèle au contexte du Web sémantique (avec l'utilisation croissant des services Web sémantiques), et de faciliter les échanges et les réutilisations d'informations répertoriées dans des registres distants.

OntoWS est composé d'un ensemble d'ontologies reprenant les spécifications décrites dans notre modèle de représentation.

L'utilisation d'ontologies pour formaliser des modèles de représentation de services Web a déjà été réalisée par [Ben Mokhtar *et al.*, 2005b] et [Suraci *et al.*, 2007] pour les raisons suivantes :

- Les ontologies rendent possible **le partage de connaissances** dans les systèmes ouverts et dynamiques (utile dans le cadre de la publication de services Web).
- Les ontologies à l'aide de sémantiques déclaratives bien définies permettent de **mettre en œuvre des raisonnements efficaces** (utiles dans le cadre de la recherche et sélection de services Web).
- Les ontologies rendent possible **l'interopérabilité des services** et fournissent des moyens pour **lier les services afin de collaborer de manière non ambiguë** (utile dans le cadre de la composition de services Web).

Les ontologies composant OntoWS sont réalisées par le fournisseur de services Web. Nous présentons tout d'abord les trois ontologies composant OntoWS (ontologies du service, du domaine d'application et du contexte d'utilisation), puis les liens qui les unissent.

OntoWS est basé sur les langages standard de description d'ontologies (OWL – [McGuiness *et al.*, 2004]. Afin de construire OntoWS, nous avons utilisé le logiciel Protégé [Knublauch *et al.*, 2004]. Chaque sous-section comporte une représentation graphique de l'ontologie (réalisée à l'aide du *plug-in* Jambalaya [Storey *et al.*, 2002]) et un exemple de fichier OWL l'illustrant[82].

82 L'ensemble des ontologies composant OntoWS est disponible à l'adresse suivante : http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/OntoWS

Ontologie du service

L'ontologie du service est composée de sept classes (*cf.* Fig. 1). Ces classes rassemblent les informations qui permettent aux clients, d'une part, d'utiliser le service (classes issues des packages *Service* et *Functional Profile* du modèle de représentation de services Web), et, d'autre part, de rechercher et sélectionner de manière efficace un service (classes issues du package *Non Functional Profile* du modèle de représentation de services Web).

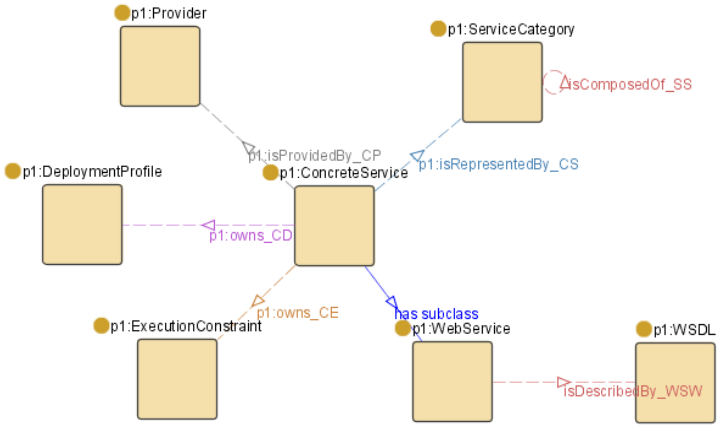


Fig. 1 Illustration de l'ontologie du service.

Les classes pour l'utilisation du service. L'ontologie du service possède quatre classes déterminant l'utilisation du service : les classes *ConcreteService*, *ServiceCategory* et *WebService* et *WSDL*.

- **La classe *ConcreteService*** est le point central de l'ontologie du service. Elle représente le service publié. Un service concret est identifié par son nom et doit appartenir à au moins une catégorie de services (propriété *isRepresentedBy_CS*).
- **La classe *ServiceCategory*** permet de décrire la catégorie à laquelle le service appartient. L'ontologie du service autorise la description de sous-catégories (propriété *isComposedOf_SS*).
- **La classe *WebService*** étend la classe *ConcreteService* et représente les services Web.
- **La classe *WSDL*** représente la description WSDL de chaque service Web (propriété *isDescribedBy_WSW*).

Pour illustrer l'utilisation de ces quatre classes de l'ontologie du service, considérons un fournisseur de service qui désire publier un service Web nommé *GeoIPService*, dont l'objectif est de localiser les utilisateurs. Ce service est une instance de la classe *WebService* et est associé à l'instance *Localization* de la classe *ServiceCategory*.

Les classes pour faciliter la rechercher et la sélection de services. L'ontologie du service possède trois classes qui décrivent l'utilisation du service : les classes *Provider*, *DeploymentProfile* et *ExecutionConstraint*.

- **La classe *Provider*** est composée de la description du fournisseur de service. Cette classe est associée à la classe *ConcreteService* à l'aide de la propriété *isProvidedBy_CP*.

- **La classe *DeploymentProfile*** rassemble l'information qui peut guider le choix des clients sur un service, telle que le prix du service, son temps de réponse et son taux de réponse. Cette classe est associée à la classe *ConcreteService* à l'aide de la propriété *owns_CD*.
- **La classe *ExecutionConstraint*** décrit les contraintes d'exécution de services publiés. Cette classe est associée à la classe *ConcreteService* à l'aide de la propriété *owns_CE*.

À l'exception des classes *WebService* et *WSDL*, spécifiques aux services issus de la technologie des services Web, les autres classes peuvent être utilisées pour décrire toutes sortes de services (Web ou non).

L'ontologie de service est utile aux fournisseurs de services pour améliorer la représentation des services à publier. Dans l'exemple qui suit (*cf.* Fig. 2), le service *GlobalWeather* est publié à l'aide de cette ontologie, réalisée par son fournisseur.

Le service Web *GlobalWeather* est décrit par son fichier WSDL (*cf.* lignes 3 à 7 pour la description de l'instance de la classe WSDL et ligne 14 pour l'association entre cette instance et celle du service Web). *GlobalWeather* est fourni par le fournisseur *webserviceX.net* (cf. lignes 8 à 11 pour la description de l'instance de la classe *Provider* et ligne 16 pour l'association entre cette instance et celle du service Web). Enfin, *GlobalWeather* est associé à la catégorie de services *Localization* (cf. ligne 12 pour la description de l'instance de la classe *ServiceCategory* et ligne 15 pour l'association entre cette instance et celle du service Web).

```
1  <rdf:RDF ... >
2      (...)
3    <WSDL rdf:ID="WSDL_GlobalWeather">
4      <url_wsdl rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
5          http://webservicex.com/globalweather.asmx
6      </url_wsdl>
7    </WSDL>
8    <Provider rdf:ID="webserviceX.net">
9      <url rdf:datatype="http://www.w3.org/2001/XMLSchema#string"></url>
10     <name_Provider rdf:datatype="http://www.w3.org/2001/XMLSchema#string"></name_Provider>
11   </Provider>
12   <ServiceCategory rdf:ID="Localization"/>
13   <WebService rdf:ID="GlobalWeather">
14     <isDescribedBy rdf:resource="#WSDL_GlobalWeather"/>
15     <isRepresentedBy_CS rdf:resource="#Localization"/>
16     <isProvidedBy_CP rdf:resource="#webserviceX.net"/>
17   </WebService>
18 </rdf:RDF>
```

Fig. 2 Extrait de l'ontologie de service du service Web *GlobalWeather* publié par son fournisseur.

Ontologie du domaine d'application

La seconde ontologie composant OntoWS est l'ontologie du domaine d'application (*ApplicationDomain*). Ceci correspond au package *ApplicationDomain* du modèle de représentation de services Web. L'ontologie du domaine d'application décrit le domaine d'application pour lequel le service peut être utilisé. Cette ontologie est composée de trois classes (*cf.* Fig. 3) : *Domain*, *Application*, *Funtcionality*.

La classe *Domain*. Cette classe décrit le domaine d'application auquel appartient le service publié.

La classe *Application*. Cette classe représente les applications associées au domaine précédemment décrit (propriété *belongsTo_AD*).

La classe *Functionality*. Cette classe décrit les fonctionnalités réalisées par les applications (propriété *isComposedOf_AF*). Cette classe montre les fonctionnalités qui doivent être implémentée afin de mettre en œuvre une application particulière. Si une fonctionnalité est complexe, elle peut être décomposée en sous-fonctionnalités (propriété *isComposedOf-FF*).

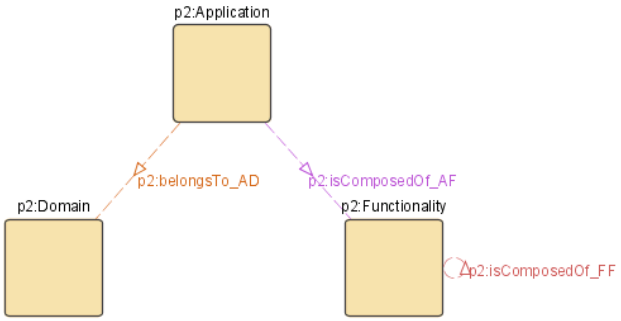


Fig. 3 Illustration de l'ontologie du domaine d'application.

L'ontologie du domaine d'application est utilisée par le fournisseur pour enrichir la représentation du service lors de sa publication et par le client pour enrichir sa requête de service lors de l'étape de recherche. Dans l'extrait de l'ontologie du domaine d'application qui suit (*cf.* Fig. 4), un client souhaite rechercher des services Web afin d'implémenter une application (nommée *NavigationAppli*, *cf.* ligne 5) appartenant au domaine de la géomatique (*cf.* ligne 7) qui inclut une fonctionnalité de localisation de l'utilisateur (*cf.* ligne 10).

```
1  <rdf:RDF (...)>
2    (...)
3    <owl:AllDifferent>
4      <owl:distinctMembers rdf:parseType="Collection">
5        <p2:Application rdf:ID="NavigationAppli">
6          <p2:belongsTo_AD>
7            <p2:Domain rdf:ID="Geomatic"/>
8          </p2:belongsTo_AD>
9          <p2:isComposedOf_AF>
10           <p2:Functionality rdf:ID="Localize_User"/>
11         </p2:isComposedOf_AF>
12       </p2:Application>
13     </owl:distinctMembers>
14   </owl:AllDifferent>
15 </rdf:RDF>
```

Fig. 4 Extrait de l'ontologie du domaine d'application représentant le domaine de la géomatique ainsi qu'une application (*NavigationAppli*) et une fonctionnalité (*Localize_User*) correspondantes.

Ontologie du contexte d'utilisation

Dans OntoWS, l'ontologie du contexte d'utilisation permet d'ajouter le concept d'*adaptation au contexte d'utilisation* à la représentation des services (*cf.* Fig. 5).

Les deux classes de base de l'ontologie du contexte d'utilisation sont les classes *ContextDescription* et *ContextElement*. La description du contexte d'utilisation est composée de différents éléments du contexte (*cf.* les propriétés entre les classes *ContextDescription* et *ContextElement*).

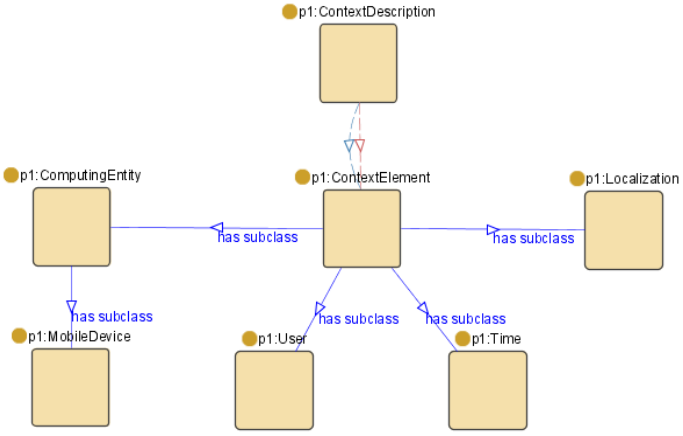


Fig. 5 Illustration de l'ontologie du contexte d'utilisation.

Dans notre travail, le contexte d'utilisation est composé de quatre éléments principaux : l'utilisateur, le temps, les entités informatiques et la localisation. Chacun de ces éléments est représenté par une classe (respectivement les classes *User*, *Time*, *ComputingEntity* et *Localization*) qui étend la classe *ContextElement*. La classe *MobileDevice* étend la classe *ComputingEntity*.

L'ontologie du contexte d'utilisation permet, d'une part, aux fournisseurs de décrire, si besoin est, le contexte d'utilisation auquel s'adaptent ses services, et, d'autre part, aux clients de décrire leur requête en terme d'adaptation. Dans l'extrait de l'ontologie du contexte d'utilisation qui suit (*cf.* Fig. 6), un fournisseur décrit le contexte d'utilisation auquel le service publié est adapté.

```
1  <rdf:RDF (...) >
2    (...)
3    <owl:DatatypeProperty rdf:ID="model">
4      <rdfs:domain rdf:resource="#MobileDevice"/>
5    </owl:DatatypeProperty>
6    <owl:DatatypeProperty rdf:ID="vendor">
7      <rdfs:domain rdf:resource="#MobileDevice"/>
8    </owl:DatatypeProperty>
9    <owl:DatatypeProperty rdf:ID="screenSize">
10     <rdfs:domain rdf:resource="#MobileDevice"/>
11   </owl:DatatypeProperty>
12   <owl:DatatypeProperty rdf:ID="ccppUrl">
13     <rdfs:domain rdf:resource="#MobileDevice"/>
14   </owl:DatatypeProperty>
15   <MobileDevice rdf:ID="MyMobile">
16     <screenSize rdf:datatype="http://www.w3.org/2001/XMLSchema#string">240x320</screenSize>
17     <ccppUrl rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
18         http://nds.nokia.com/uaprof/NN95_8GB-1r100.xml
19     </ccppUrl>
20     <model rdf:datatype="http://www.w3.org/2001/XMLSchema#string">N95_8GB-1</model>
21     <vendor rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Nokia</vendor>
22   </MobileDevice>
23 </rdf:RDF>
```

Fig. 6 Extrait de l'ontologie du contexte d'utilisation représentant le dispositif mobile pour lequel le service publié apporte une réponse adaptée.

La classe *MobileDevice* est composée de quatre propriétés : le modèle du dispositif d'accès – `model` (*cf.* lignes 3 à 5), le fabricant – `vendor` (*cf.* lignes 6 à 8), la taille d'écran – `screenSize` (*cf.* lignes 9 à 11) et la localisation du profil du dispositif – `ccppUrl` (lignes 12 à 14). Dans l'exemple, le fournisseur décrit un contexte d'utilisation (dispositif Nokia N95 – lignes 20 et 21, dont la taille d'écran est de 240x320 pixels – ligne 16) auquel le service publié est adapté.

Ontologie OntoWS

OntoWS intègre les trois ontologies précédemment étudiées (les ontologies du service, du domaine d'application et du contexte d'utilisation) et construit des liens entre ces ontologies (*cf.* Fig. 7). Nous décrivons tout d'abord les propriétés entre les ontologies du service et du domaine d'application, puis les propriétés entre les ontologies du service et du contexte d'utilisation.

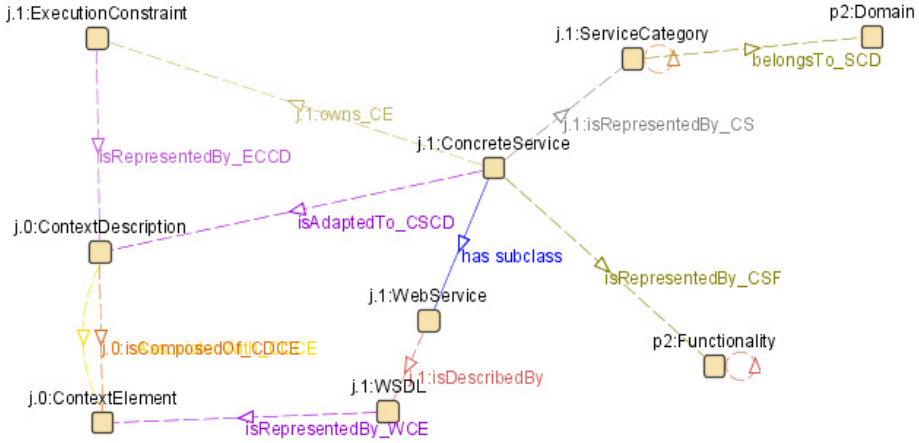


Fig. 7 Illustration d'un extrait de OntoWS représentant les classes et les propriétés mises en jeu pour formaliser les relations entre les ontologies (seules les classes et les propriétés participant aux relations entres les ontologies sont représentés dans cette figure).

Les propriétés entre les ontologies du service et du domaine d'application. Deux propriétés lient ces deux ontologies.

- **La propriété *belongsTo_SCD*** relie les classes *ServiceCategory* et *Domain* afin de représenter la catégorie de services à laquelle appartient le service décrit.
- **La propriété *isRepresentedBy_CSF*** relie les classes *ConcreteService* et *Functionality* afin de permettre aux clients de connaître les tâches que les services implémentent.

Dans l'extrait d'OntoWS qui suit (*cf.* Fig. 8), un fournisseur associe au service qu'il publie nommé `Localize` (ligne 3), la tâche `Localize_User` présente dans l'ontologie du domaine d'application (lignes 4 et 5).

```
1  <rdf:RDF (...) >
2    (...)
3    <j.1:ConcreteService rdf:ID="Localize">
4      <j.1:isRepresentedBy_CS
5      rdf:resource="http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/OntoWS/Services.owl#Localization"/>
6      <isRepresentedBy_CSF
7      rdf:resource="http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/OntoWS/ApplicationDomain#Localize_User"/>
8    </j.1:ConcreteService>
9  </rdf:RDF>
```

Fig. 8 Extrait de code de OntoWS représentant la fonctionnalité qu'implémente le service *Localize*.

Les propriétés entre les ontologies du service et du contexte d'utilisation. Trois propriétés entre ces deux ontologies permettent d'ajouter un niveau de sémantique à la représentation de services en termes d'adaptation au contexte d'utilisation.

- **La propriété *isAdaptedTo_CSCD*** relie les classes *ConcreteService* et *ContextDescription* afin d'exprimer le contexte d'utilisation pour lequel le service apporte une forme d'adaptation.

- **La propriété *isRepresentedBy_ECCD*** relie les classes *ExecutionConstraint* et *ContextDescription* pour représenter des contraintes d'exécution spécifiques au contexte d'utilisation.
- **La propriété *isRepresentedBy_WCE*** relie les classes *WSDL* et *ContextElement* afin d'ajouter une information supplémentaire à la description fonctionnelle fournie par le standard WSDL.

Dans l'extrait d'OntoWS qui suit (*cf.* Fig. 9), un fournisseur ajoute la description du contexte pour lequel son service, nommé `MyService`, est adapté. Dans cet exemple, le service est adapté à la description du contexte associé à l'élément `MyMobile`, défini dans l'ontologie du contexte d'utilisation (lignes 5 à 10).

```
1  <rdf:RDF (...)>
2    ...
3    <j.0:ConcreteService rdf:ID="MyService">
4      <isAdaptedTo_CSCD>
5        <j.1:ContextDescription rdf:ID="MyContextDescription">
6          <j.1:isAssociatedWith_CDCE
7          rdf:resource="http://www-lsr.imag.fr/users/Celine.Lopez-Velasco/OntoWS/UseContext.owl#MyMobile"/>
8        </j.1:ContextDescription>
9      </isAdaptedTo_CSCD>
10   </j.0:ConcreteService>
11 </rdf:RDF>
```

Fig. 9 Extrait de code de OntoWS représentant la description du contexte pour lequel le service *MyService* est adapté.

Les extraits d'OntoWS étudiés précédemment ont montré que OntoWS peut être utilisé afin de compléter la représentation des services publiés par les fournisseurs. Cependant, OntoWS permet aussi de faciliter l'étape de sélection de services réalisée par les clients. Des langages tels que SPARQL [Prud'hommeaux *et al.*, 2008], SWRL [Horrocks *et al.*, 2004], permettent d'extraire facilement les informations à partir des différentes ontologies.

Annexe 2 : Signatures des méthodes des services Web d'interface de WSR

WSR possède trois services Web proposant chacun un type de fonctionnalité : un service Web d'interrogation, un service Web de publication, et un service Web de recherche. Les signatures de chacune des méthodes de ces services sont listées ci-dessous (la description des méthodes est faite dans le chapitre portant sur WSR).

Services Web d'interrogation. Ce service Web contient cinq méthodes. Leurs signatures sont les suivantes :

- Rdf **CategoryRequest**()
- Rdf **FunctionalityRequest**()
- Rdf **FunctionailtyRequestByDomain**(String Domain)
- Rdf **UseContextRequest**()
- Rdf **UseContextElementRequest**(String elementName)

Services Web de publication. Ce service Web contient six méthodes. Leurs signatures sont les suivantes :

- Integer **AddFunctionalDescription** (String name, Url wsdlLocalization, Float price, String comment)
- **AddNonFunctionalDescription** (Integer serviceKey, List associatedCategory, List associatedFunctionality)
- **AddExecutionConstraintDescription** (Integer serviceKey, String elementName, List attributes)
- **AddUseContextDescription** (Integer serviceKey, String elementName, List attributes)
- Boolean **CategoryRecord** (String categoryKey, String description, String categoryParentName)
- Boolean **FunctionalityRecord** (String functionalityKey, String description, String functionalityParentName)

Services Web de recherche. Ce service Web contient trois méthodes. Leurs signatures sont les suivantes :

- List **byKeywordSearch** (String keyWord)
- List **byCategorySearch** (String categoryName)
- List **conceptionSearch** (List functionality, List useContext)

Annexe 3 : Signatures des méthodes des services Web d'interface de GenAWS

GenAWS possède deux services Web d'interface. Le premier service Web permet d'instancier de nouvelles définitions de composition dans la base de connaissances méthodologique. Le second service Web permet la génération d'applications adaptées au contexte d'utilisation.

Service Web d'instanciation de la base de connaissances méthodologique. Ce service possède une méthode. Sa signature est la suivante :

- `Boolean` **`RecordWSComposition`** `(String WSCName, URL WSCDefinition, List WSCategory, List fonctionnality)`

Service Web de génération d'applications adaptées au contexte d'utilisation. Ce service possède deux méthodes. Leurs signatures sont les suivantes :

- `URL` **`AppliGenDef`** `(URL ProbCWSMainTask)`
- `Map` **`AppliGenExec`** `(URL ProbCWS, Map inputsValue)`

RÉSUMÉ : Ce travail se situe dans le domaine de la conception des applications à base d'architecture orientée services adaptées au contexte d'utilisation. Ce type d'architecture permet les échanges entre les fournisseurs de services et les clients qui conçoivent de telles applications. Afin que les clients trouvent le service correspondant aux mieux à leurs besoins, les services doivent être décrits par leur fournisseur selon un processus standard. Ceci permet à ces services d'être réutilisés, découverts et composés. La combinaison de ces services doit apporter un résultat adapté au contexte d'utilisation (l'utilisateur, la localisation, le temps, et le dispositif utilisé). Nous proposons dans ce travail une solution qui englobe les processus de description, de recherche, et de composition de services, en ajoutant de manière transversale l'adaptation au contexte d'utilisation.

Le standard de description de services Web WSDL ne permet qu'une représentation des aspects fonctionnels des services (méthode, paramètres échangés, et protocole d'accès). Afin de faciliter les étapes de recherche et de sélection effectuées par les clients, la représentation de services doit être enrichie des aspects liés au domaine d'application auquel les services sont dédiés, les aspects non fonctionnels (tels que la description du fournisseur et des contraintes d'exécution) et le contexte d'utilisation auquel les services s'adaptent. Le modèle de représentation de services Web proposé, appelé WSR-Model, fournit l'ensemble de ces catégories d'informations afin que les fournisseurs publient leurs services et que les clients réalisent les étapes de recherche et de sélection. Ce modèle est opérationnalisé via le système de représentation de connaissances par objets AROM qui implémente le registre de services Web, que nous nommons WSR.

La composition de services Web repose sur une description de la planification des services et sur l'exécution de cette planification. À ce jour, les solutions existantes ne prennent pas en compte l'évolutivité de la composition et la prise en compte de l'adaptation au contexte d'utilisation lors des phases de description et d'exécution de la composition. Nous proposons un modèle de composition de services Web, appelé ProbCWS, qui s'appuie sur les méthodes de résolution de problèmes à base de modèle de tâches. La définition de la composition est définie comme un problème à résoudre, dont les tâches de résolution de plus faible granularité sont des services Web. La plate-forme de génération d'applications adaptées, nommée GenAWS et intégrant ProbCWS, fournit aux clients un moyen de composer à la volée des applications adaptées. La mise en œuvre de GenAWS est réalisée, entre autres, par le langage de résolution de problèmes AROMTasks, sous-jacent au système AROM.

MOTS-CLÉS : Architecture Orientée Services, Services Web, Adaptation, Contexte d'utilisation.

ABSTRACT: This thesis concerns the domain of applications adapted to use context issued from the field of Service Oriented Architecture. This type of architecture enhances the cooperation between service providers and service consumers that design SOA applications. In order to help consumers, finding the service that best suits theirs needs, the services must be described by the provider following a given description standard. This description allows services to be discovered, reused and integrated in new applications. The composition of several services must yield results adapted to the use context (user, localization, time, access device). We propose in this work a solution that encloses the description, the retrieval and the composition of services and that takes into account at all levels the adaptation to the use context.

The standard for Web service description, WSDL, concerns only the functional aspects of services (methods, parameters, access protocol). For facilitating the retrieval and the selection, the representation of Web Services has to be enriched with information related to the application domains, to the non-functional aspects (such as, description of the provider, execution constraints) and to the use context for which the service was conceived. The model for Web Service representation, called WSR-Model, covers the above information categories in order to enhance the publication, the retrieval and the selection of Web service. This model is supported by a register called WSR, which was implemented using AROM, an object based knowledge representation system.

The composition of Web services is based on the description of the execution plan of a collection of services. Nowadays, the existing solutions do not take into account the evolution of such compositions nor the adaptation to the use context. We propose a composition model, called ProbCWS, which transposes the problem solving methods based on task models to Web service compositions. A composition is defined as a decomposable problem for which the fine-grain solving tasks are Web services. The generation framework, called GenAWS, integrates ProbCWS and gives consumers a mean to compose on the fly adapted applications. GenAWS takes advantage of AROMTasks, a problem solving language that is part of AROM framework.

KEYWORDS: Service Oriented Architecture, Web Services, Adaptation, Use context.